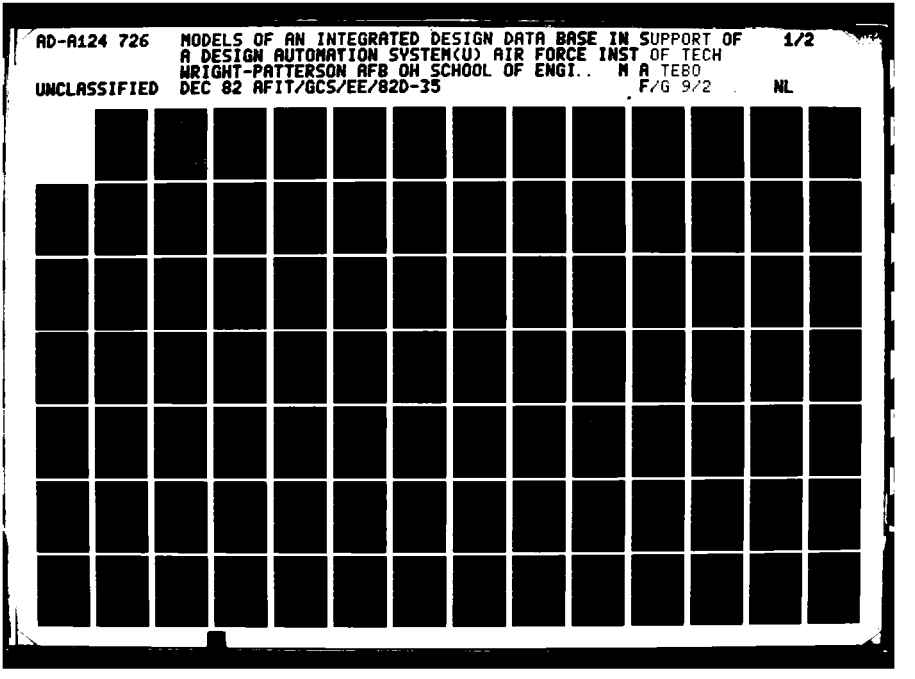
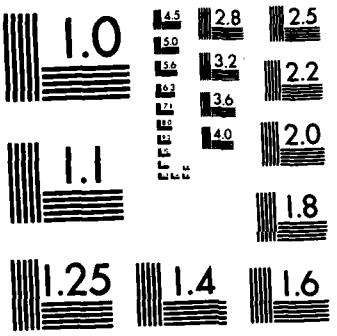

AD-A124 726
MODELS OF AN INTEGRATED DESIGN DATA BASE IN SUPPORT OF A DESIGN AUTOMATION SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M A TEBO
1/2
UNCLASSIFIED
DEC 82 AFIT/GCS/EE/82D-35
F/G 9/2
NL



DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/82D-35

MODELS OF AN INTEGRATED DESIGN DATA BASE IN SUPPORT OF A DESIGN AUTOMATION SYSTEM

THESIS

AFIT/GCS/EE/82D-35 Michael A. Tebo
Captain USAF

Approved for public release; distribution unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFIT/GCS/EE/82D-35	2. GOVT ACCESSION NO. A124 726 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): MODELS OF AN INTEGRATED DESIGN DATA BASE IN SUPPORT OF A DESIGN AUTOMATION SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: MS Thesis
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Michael A. Tebo, Captain, USAF	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT/EE) Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: December 1982
	13. NUMBER OF PAGES: 172
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE; IAW AFR 190-17

19 JAN 1983

Approved for public release: IAW AFR 190-17.

LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AIC)
Wright-Patterson AFB OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CANONICAL SCHEMA
COMPUTER AIDED DESIGN
CONCEPTUAL SCHEMA
DATA BASE
DESIGN AUTOMATION
MICROELECTRONIC DESIGN

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents a model for an integrated design data base that will be used within an integrated design system for microelectronics. This report also describes a model of an integrated design system, whose functions include a single, flexible interface between the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

designer and the design system. These two models provide a conceptual-level design of a Design Automation System; however, the emphasis in this report is on the data base model.

The results described in this report are two of the models necessary for the design of a Design Automation (DA) System. The first model is a high-level design which shows the components and the interactions of these components within the DA System. The second model constitutes the design at the conceptual-level of the data base required by the DA System. This data base, called an Integrated Design Data Base, is an integral part of the DA System. The model of the data base defines the data requirements of the design tasks within the microelectronic design cycle.

AFIT/GCS/EE/82D-35

MODELS OF AN INTEGRATED DESIGN DATA BASE IN SUPPORT OF A DESIGN AUTOMATION SYSTEM

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Michael A. Tebo, B.S.
Captain USAF

Graduate Electrical Engineering

December 1982

Preface

After having worked for three and a half years in a Computer-Aided Design Facility for microelectronics, many of the problems with the design tools available to the designer became apparent. Most of all the complaints and frustrations were caused by the ineffective computer use by the design tools. The two worst problems were the inflexible user interfaces to the computer and the many data bases that had to be maintained during the design cycle. At the Air Force Institute of Technology the exciting opportunity to address both of these problems was presented by Dr. Carter. He proposed a plan to design and implement a Digital System Design Automation Facility. This thesis project is one of the many efforts required to bring this plan to completion.

Special thanks are due to Tom Herbert, Gary Pritchard, and J. B. Rawlings who patiently helped me during my developmental years at the Avionics Laboratory. Thanks are also extended to Rick Stormont and Bill Ure for their time and aid. Also the patient and insightful help provided by Dr. Carter was greatly appreciated.

A thank you to my parents for their sacrifices and lessons that I will never forget. Finally, a warm and

special thank you to my fiancee, Chris, for her patience, understanding, and support during this long educational journey.

— Michael A. Tebo

Contents

List of Figures

Abstract

This report presents a model for an integrated design data base that will be used within an integrated design system for microelectronics. This report also describes a model of an integrated design system, whose functions include a single, flexible interface between the designer and the design system. These two models provide a conceptual-level design of a Design Automation System; however, the emphasis in this report is on the data base model.

The results described in this report are two of the models necessary for the design of a Design Automation (DA) System. The first model is a high-level design which shows the components and the interactions of these components within the DA System. The second model constitutes the design at the conceptual-level of the data base required by the DA System. This data base, called an Integrated Design Data Base, is an integral part of the DA System. The model of the data base defines the data requirements of the design tasks within the microelectronic design cycle.

CHAPTER I

INTRODUCTION

Introduction

The challenge addressed in this report is to design models of a Design Automation System (DA System) including its integrated data base. This model development is a first step in designing a tool that will be used in the design of microelectronic devices (ICs, PCBs, Hybrids). This tool (DA System) has often been designed and implemented separately with largely unsuccessful results. Instead, there must be a coordinated effort, integrating the designs of both the DA System and its data base. This coordinated design is necessary because there is a symbiotic relationship between the DA System and the data base. Therefore, decisions must be considered with both the data base and the DA System taken into account. "The data base is needed to tie all the elements of the design together and to allow automation. The design must be integrated into the CAD data base [6:550]." As used in this report, the data base embedded in the DA System will be called an Integrated Design Data Base (IDDB), because it is an Integrated (into the DA System) Design (type of data contents) Data Base (IDDB).

The design of the DA System including its IDDB described in this report will support hierarchical design without a prespecified selection of design representations. The design of the DA System and the IDDB models is founded on current Software Engineering and Data Base Design Techniques. The objective guiding the design was to concentrate on providing an interface to design data and to design tools that would improve the efficiency and effectiveness of the designer's work.

Important Concepts

Independence

Data Independence. The object of data independence is to clearly differentiate between the logical and physical aspects of data base management. These differences include data base design (total system), data retrieval (physical), and data manipulation (logical). The major advantage of data independence in model design work is that it provides a protective "buffer" against damage, or data integrity problems, which are caused by growth and restructuring of the data base (27:106). Growth and restructuring are the components that force disconcerting changes in a data base. Two of these buffers are required, in both a data base model as well as a DA system model. Each buffer will protect against the major components of data integrity problems; i.e., growth and restructuring. These two

buffers are defined to be physical and logical data independence.

Physical Data Independence. This term implies data and program immunity to changes in the physical storage structure. This means that the physical storage organization and its implementation may be changed without changing either the logical structure of the data or the Application Program's data concerns. Other authors call this term "Hardware Independence."

Logical Data Independence. This term implies data and program immunity to changes in the data model. This means that the total logical structure of the data may be changed, but these changes will not affect the Application Programs (AP). Other authors call this term "Software Independence."

Schema

Conceptual Schema. This term represents the entire data base as an abstract model description that provides a mapping function between the logical and physical schemas. The conceptual schema is the framework which will hold the values of the data items. This data base model, the conceptual schema, is independent of hardware and software considerations. Therefore, the conceptual schema is both physically and logically independent. Data independence

is required to keep the conceptual schema as stable and as long-lasting as possible. Stability is needed in the design because the conceptual schema is the foundation upon which the rest of the design is built. The other two components of the data base model, physical and logical schemas can and will change, but the conceptual schema should not change. Instead, it is designed to be resistant to change, if its two buffers work properly.

Physical Schema. This is the storage organization of the data base including its implementation on some storage media. Storage mechanisms and access methods are elements of the physical schema. The physical data structure is the form in which the data is recorded on the storage media. This characteristic of the data base model provides the physical data independence required by the canonical schema.

Logical Schema. This term is a description of the data as seen by the users of the data base. Each user, or each design task, has its own view of the data (called a task-view in this report, sub-schema by other authors) which consists of definitions of each of the data items, their relationships, and the format of the data as seen by the design task. The logical data structure is the structure of the data as required by a design task. This

representation of the data base model provides the logical data independence required by the canonical schema.

Considerations. The need to create an integrated design system with the data base and application programs integrated into the system has been generated by microelectronic design complexity design tool incompatibility, and the designer's need for automated design tools that are easily and effectively used. LSI/VLSI/VHSI circuits must be designed quicker, cheaper, reliably, and more testable, despite the increased complexity of the design process.

The approach taken in this report is that VLSI/VHSI circuit design must be done using an automated design system that is composed of a set of integrated design tools and associated data bases. The integration of the data bases makes up the core of the integrated design system and provides the key to the integration of the entire DA system. The design system utilized here is called a Design Automation System. It includes an Integrated Design Data Base (IDDB), which is an essential, and in fact an inherent element of such a Data System. Thus, when DA System capabilities and functions are discussed, it is assumed that the IDDB is a part of this system.

Designing and implementing an integrated DA system is difficult because of the many functional components of such a system. These required components include Application Programs (AP), an operating system, peripheral

interfaces, interprocess communications, a Data Base Management System (DBMS), and user interfaces. However, the key component of the DA System is the Integrated Design Data Base.

> But what is possibly the most important, the various software pieces of the CAD puzzle--usually incompatible programs written at different universities in different languages running on the different operating systems of different computers--are being knitted together into cohesive operational programs drawing upon unified data bases [9:74].

The IDDB is characterized by the access and the use of a centralized data base by the design tools or APs. The data is physically stored independently of the software that will access it; and the data is accessed by the software independently of the storage medium. This data independence characteristic of the IDDB must be "designed-into" the model from the beginning. Furthermore, it is essential for the model to be physically and logically independent.

Design Considerations

There are three major views of a data base design, each of which must be designed in turn. These three views are called the conceptual, logical, and physical views or schemas of the data base. This report describes a conceptual-level model, conceptual schema, of an IDDB.

The DA System will be designed similarly to its IDDB. Because of integrated design decisions that must

be made, the DA System and IDDB must be designed in concert. The DA System must be designed at the conceptual-, logical-, and physical-level before implementation. This report also includes a description of a conceptual-level model of a DA System.

The first and most important element of the design process is the conceptual-level design.

> The CAD data base system [has] two major design goals: to unify existing CAD tools and to provide a nucleus for future growth. To meet these goals, a data base structure [more properly, a data base SCHEMA that defines such a structure] has to be defined. . . [22:401].

It is especially important in this report because of its emphasis on the model design of the IDDB. The conceptual schema design will describe the integration of the data. The conceptual schema will also define the data requirements of the data base with a flexible specification of the data requirements, which will allow the design to absorb changes and growth to the data base. The

> . . . conceptual model [conceptual schema] plays a crucial role in the data base design process: first, it provides the framework within which user requirements must be identified and understood. Second, the model [conceptual model] provides a specification mechanism for communicating the global conceptual view of the enterprise [11:546].

Purpose

The purpose of this report is to present the conceptual-level model design of the DA System and its IDDB. The DA System model will be described in relatively

general terms, with its component parts, and the data and control paths defined. These component parts include the Executive, DBMS, APs, and Data Dictionary. The IDDB model will be described in greater detail. A complete conceptual schema model of the IDDB will be developed and explained. All of the data requirements and the organization of the data will be described through the design of the conceptual schema.

The two major components of this report are the IDDB and the DA System. They each have a symbiotic relationship to each other. The DA System is the environment for the Integrated Design Data Base. Thus, the data base is dependent on the design of the DA System. However, because of the dynamic nature of the DA System's application programs, the DA System demands that the data base which supports the APs must be physically and logically independent. This explains the DA System and the IDDB interdependence. The author maintains, that to adequately describe the DA System and its IDDB, both must be designed in concert.

Another reason to design an IDDB and a DA System together is that VLSI/VHSI circuit design demands a DA System with an integrated design data base. The literature (10:353) clearly shows that with today's design complexities, automated design cannot be done by the "bag of tools" approach. Present designs are so complex that structured

engineering approaches are being used in design work. These same structured approaches must be engineered into the design tools (AP) and techniques used by the designer.

> The effective use of these computer programs necessitates structured design applications so that the complexity of the design and verification tasks is reduced to a manageable level. Large amounts of data and a variety of design representations are used for each circuit. Thus, it is important that an integrated set of computer aids, coupled with a unified approach to data management, be provided to the IC designer [12:1197].

Once the present design tool problems ("bag of tools" approach), are solved, the designers can return to the true microelectronics design problems, because a usable set of integrated tools facilitating design work will have been developed.

Design tools are needed, which include a DA System, IDDB, and application programs, that can be easily used by designers, and have no inherent problems of their own. These tools should be just as easy and as flexible for the designer to use as a paint brush or a hammer are to a painter or a carpenter. The designer should not be aware of the data base, the DA System, or the individual requirements of these design tools. Instead, the designer should only be faced with the problem of the design itself. To provide such design tools, the DA System and its data base, must be transparent to the designer, and perform all tasks, quietly and responsively.

Goals

The goals of this thesis are to:

1. Provide the reader with a clear understanding of both the background of the problem and its solution. In providing this description, the characteristics of the problem that cause the greatest design difficulties will be described, along with a discussion of the major factors that are essential in creating a solution.

2. Describe the software engineering and data base design techniques and the design approach used during the design efforts.

3. Present important, known data requirements for the design tasks to be performed by the DA System, and characterize these design tasks.

4. Conceptually design the DA System, and use this design as a model to describe the environment of the IDDB. Insure the DA System and the IDDB designs are physically and logically independent of hardware and software considerations.

5. Conceptually design the Integrated Design Data Base and describe the data contents and the data relationships within the data base.

6. Provide recommendations concerning the implementation and the maintenance of the IDDB.

Overview

As previously stated, the purpose of this report is to provide a conceptual-level model of the DA System and its Integrated Design Data Base. The first three chapters provide the essential introductory information concerning this report's design efforts. Chapter II discusses the important background elements of both the design challenges (problems) and the solutions to these challenges. Once the reader has a grasp of the important characteristics of the background, Chapter III provides an explanation of the software engineering and data base design techniques and the design approach that was used.

The DA System model is presented in Chapter III. This design is presented in two parts: (1) the organization, and (2) the usage of the DA System. Chapter V presents the conceptual-level model of the IDDB. This design consists of an explanation of the individual relationships of the data, the information content of these relationships, and the total system view of this representation of the IDDB.

Conclusions are given in Chapter VI that discuss how the six goals outlined above were met by the work described in this report. Functions of the Data Base Administrator (DBA) along with important characteristics of the data base models and a DBMS are discussed in

Chapter VII. Recommendations are also provided in Chapter VII which describe the implementation of the DA System including its IDDB.

CHAPTER II

BACKGROUND

Introduction

In a broad sense, the Age of Microelectronics has arrived. New ways are being discovered daily to use microelectronic devices in unique ways. In addition, greater demands are being placed on the performance characteristics of microelectronic devices. These additional uses and demands have an effect on the design of these devices. The effect is that the designs are becoming more complex which causes the design problems to increase.

The designer has to bear the additional burdens created by the increased complexity of the microelectronic design. In the past the designer developed the need for design tools to help in the design process. Out of necessity, the designer is being aided by the use of computers. The purpose of design automation computer programs is to minimize the complexity of the design tasks for the designer. Computer use by designers is a source of help. Unfortunately, it is also a source of problems. Computer use certainly helps reduce the complexity of the design work, but the computer is not an easy design tool because of inflexibility in responding to different designer requirements.

The objective of this report is to improve the computer's usability for the designer's work on microelectronic design. The problems to be addressed in this report are some of the factors that negatively affect the computer's usability. These problems must be overcome if truly effective use of computer-aided design tools is to be realized.

Background to the Problem

The problems are basically generated from two sources which are characteristics of:

1. the microelectronic design environment, and
2. the designer.

This background text will describe the basic problem areas of these two sources. It should be remembered that characteristics of these two problems are interrelated.

As a prefacing remark, it should be noted here that there is a difference between "incorporating" versus "integrating" software tools or data bases into a design system. "Incorporating" implies that the tools and data bases are gathered and are available for a designer's use; in effect, a "bag of tools" approach. CAD Systems are usually nothing more than this, containing diverse data bases with a different data base for each application program. "Integrating" implies that there is an integrated design data base, where all of the design tools access one data base and the boundaries between individual programs are "fuzzy." The boundaries are "fuzzy" because designers

do not have to re-submit the entire design description data for each Application Program (AP) used. The majority of the design description has already been supplied from previous AP runs. There is only a small amount of additional data required to augment the present design. For a DA System to function as described here it must have integrated data bases as well as integrated design tools.

Growth

The microelectronic design field has grown rapidly; consequently, the design tools and techniques are changing. The diversity of the design problems and the complexity of the solutions to these design problems have grown. The knowledge required and skills required for the designer to perform an effective job using the new design tools and techniques have also increased dramatically.

This growth has affected not only the designer's performance but also the design tools created to minimize the design complexities. Usable design tools that will provide solutions for the design problems are needed and are causing researchers to improve the present design tools. But the rapid pace of microelectronic changes tends to prevent researchers from designing for the future. Instead, new tool development requirements are driven by today's problems, but end up being yesterday's solutions. There are "two constants found in design automation systems--growth and change [35:464]." Not only are the tools being

developed too late, but in the past they were not being designed well (18).

Designers

Veteran designers are accustomed to having complete control and knowledge of their design work. However, it is almost impossible for one person or even one team to be aware of all the advances and potential ways to solve present design problems. This often means that viable design tools exist to be used, but the tools are not a coherent set. Furthermore, they cannot be easily concatenated into an incorporated system. Thus, use of the design tools (application programs, design systems, data base, etc.) that can help solve the design problem often introduce more problems. These new problems can be in the form of translation from one data format to another, from one computer representation to another (60 bits to 32 bits, ASCII to EBCDIC, etc.), new input/control languages, or different data base formats. An example of a designer's nightmare is controlling design data integrity, which can easily cause problems in an incorporated design system. For instance, the design may have to be manually checked to prove that the resultant design of an AP is logically and functionally equivalent to the design input to the AP.

> Incompatible data may make it impossible to determine if, say, the circuit simulated with logic simulator is in fact the same circuit simulated via the circuit simulator [2:108].

Bag of Tools Approach

New microelectronic design problems are forcing the designer to use new design tools. Often, to avoid losing personal control of the design, the designer may begin to design using the "bag of tools" approach. The typical solution is to find stable, understandable design techniques that allow personal design control, contain only a couple of input/control languages to learn, and safe data (i.e., card deck stored on desk). This solution typifies the "bag of tools" approach, also called an incorporated CAD System.

Incorporated CAD Systems

Many incorporated CAD systems are inefficient and unresponsive to the designer's needs. They require different control languages for each different Application Program. The different APs also require the design data to be reformatted for each AP. Thus, each AP must be provided all the data required in its own format, and the syntax and contents of the data must also be correct. The interface programs that were needed to convert the data format from one program's format to another were also a hinderance. So, CAD Systems tended to be either too imposing to learn how to use, or too frustrating because of the constant updates to new interfaces for the application programs.

Within the microelectronic design field there are whole companies and sciences devoted to specialized sectors of this diverse field. For each sector, design tools and related elements are developed, which include:

1. specialized software,

2. standardized component and electrical characteristic libraries,

3. design rules,

4. individual data bases for each software package, and

5. specialized input/output control command requirements.

A design system consists of a collection of these above design tools and related elements. There are huge amounts of redundancies, overhead and interface programs and data requirements that must be contended with when using a design system that only "incorporates" the design tools. These are elements of the problem to be solved and they make the computer very difficult to use for design work. Thus, an incorporated design system is difficult and inefficient to use, and the

> . . . waste [was] centered around the development and use of many translators which [are] needed to convert data files from the format of one application program to that of another. The main concern of user [designer] frustration came from the diversity of input foremats and diagnostics which forced the user to become familiar with many different languages [10:353].

Data Bases

There are data problems to be addressed that affect the usability of design systems. Designers have experienced the lack of data integrity and this is a very serious data base problem. It is very difficult for the designer to verify the integrity of the design data. Briefly, data integrity is the preservation of data items, their associations, and their values. In this case, to verify data integrity, the designers are concerned that the design data has not been lost or compromised (changed). There are two ways to perform the verification: (1) compare the data character-by-character, or (2) have an accurate data management function, which will monitor and maintain the design data. Difficulty in verifying the design data is another factor that causes the designer to find that design systems are difficult to use and inflexible.

Example

A simple example representing an incorporated CAD system with diverse data bases should show the relevant problems. Consider a car requiring mechanical repair. Experts are required to perform mechanical repairs. An expert has the special skills, knowledge, and tools that must be used to make proper repairs. An expert is usually limited to only one field. Thus, there is an incorporated group of expert mechanics that will be used to work on the

car. An expert in fan belts is required to change the fan belt. After the change is accomplished, the alternator is diagnosed as needing repair or replacement, so the alternator is replaced. The job is very difficult because the fan belt gets in the way. But the replacement is made in spite of the difficulty. The ability to replace the alternator, with the fan belt in the way, is the major skill required in alternator repair. During the process of changing the alternator, the fan belt is often moved, maybe forcing readjustment. Next, the generator must be replaced, which requires another expert. This time both the alternator and the fan belt are in the way, and both get accidentally moved. Thus, the generator replacement may force reiteration of the two previous repairs. This exemplifies how each step of a design (replace) or a verification (adjust) can have a rippling effect throughout the system (engine).

As in the preceding example, design programs and their data constantly change, and each data format or input/control language change causes a ripple, affecting all subsequent programs and data bases. During each step of the design cycle, data can be mistyped or translation programs may introduce errors. These are comparable to the accidental readjustments that the mechanics were doing on the car parts. And even more importantly, it can be seen that the major skill required is the knowledge of how to get the

job done in spite of the non-integration of skills (input/control languages) and tools (data bases). The "bag of tools" (division of tools) approach has also led to optimization and total design problems. ". . . the division has blindered designers, forcing them to attempt optimization without regard to its impact on the overall circuit [2:108]."

The redundant "specialized" skills or tools (i.e., translators), and knowledge required for each application program (seen in the above example) are easily reduced. The reduction occurs as a result of a DA system because of the integration of its design data base and AP.

> Each CAD program usually has its own unique input language, library, and data storage. The designer must take the time to learn each input language, and, even worse, must specify the particular design (often with redundancy, sometimes with error) for each program he desires to use. For example, circuit connectivity must be given in different forms for logic simulation, test generation, and circuit layout programs [22:399].

The proper implementation of an Integrated Design Data Base to be used within the context of a well-conceived DA System will provide a flexible and usable design tool solution for the designer.

Background to the Solution

The objective of the solution is to provide the design of a usable set of tools that can be applied by the designer for microelectronic design work. The designer must be able to use the tools efficiently (little wasted motion and time) and effectively (accurate design). "The

design community must perceive that use of the data automation tool will cost-effectively result in a tangible benefit [5:5]."

Introduction to Elements of the Solution

The problem background section discussed the existing problems in the absence of elements of the solution. As previously mentioned, the elements which comprise the solution are the use of Software Engineering, Data Base Design Techniques and Tools, a Design Automation System, and an Integrated Design Data Base.

Software Engineering. The author has noted a distinct lack of software engineering tools and techniques used in the design and implementation of existing CAD software (14:21,23).

> The notion of software engineering was introduced in 1968, to refer to the goal of applying traditional forms of engineering discipline to the production of software [16:43].

The cause of the software engineering deficiencies, the author believes, is because most of the software developed up until the late 70s was written by the designers who only wrote the program or software module to solve specific, parochial design problems. Because software engineering concepts are relatively new, the concepts were not used to design incorporated design systems.

Some of these inferior practices include: insufficient documentation, hardware-dependent software scattered throughout the program modules, and software modules which are strongly coupled and have loose cohesion. Usually there is no input/output interfaces to other programs, very few standardized data formats, and difficulty in interfacing most associated data bases.

Software engineering tools and techniques, while relatively new, have proliferated because of the software development industry. There is a critical need for a structured, engineering approach to software design and development. Software is much too complex to design without software "design rules." Using software engineering techniques, the resulting software has become far easier to use, debug, maintain, and comprehend. During the software development, implementation, and verification phases, some of today's software, such as DBMS and data base models, can now be designed and analyzed with mathematical rigor.

Software engineering techniques are used in the model design of the DA System and its IDDB presented in this report. Once the discussion of the design of the DA System and its IDDB is complete, software engineering techniques are used to provide ways to comprehensively view these complex designs. Thus, use of software engineering techniques allows the complexity of the design to be minimized and the informational content of the design to be maximized.

Data Base Design. An integrated design data base used by a DA System does not normally evolve out of the presently used diverse data bases existing within an incorporated CAD System. Therefore, the IDDB was carefully designed using Data Base Design techniques. These techniques are a subset of software engineering techniques. Essentially, general software engineering techniques are applied to specific data base design problems. Thus, a structured engineering approach was taken during the design of the Integrated Design Data Base model.

Integrated Design Data Base. One of the goals of an IDDB is similar to that of a DA System. The goal of a DA System is to reduce the impact of software, hardware, and data base changes to existing AP. This is similar to the IDDB goal, which is to isolate the logical and physical data structures from the AP, or logical and physical independence.

The design of the IDDB must begin with the design of the information structure as the first step. This information structure is an abstraction from the complexity of the real world into a logical structure consisting of information needed for the IDDB. The second step is to reduce the information structure into a data structure suitable for management by a DBMS (4:113). While using the IDDB, the designer's design data will not be under personal control.

However, the data integrity and redundant data problems previously experienced with diverse data bases will disappear through use of an integrated design system. The designer will find an added plus with the maintenance, update, and validation procedures that will now be available for use. Thus, using the proper software engineering techniques, and designing the DA system in concert with the IDDB, should provide designers with the design tools that they have needed.

DA Systems. As previously discussed, there are many problems concerning the usability of a DA system. Good DA Systems have been built, but the majority of these design systems described in the literature, still have designer complaints concerning usability, as previously discussed. These problem areas are improved by the work provided in this report's models. Physical and logical independence is "designed-into" the system and provides a flexible schema (data base model) design.

Air Force Institute of Technology (AFIT) Implementation Plans

The Department of Defense and the Air Force are very active in implementation requirements in the field of microelectronics, Design Automation (DA), military design for the following reasons. All major weapon and support systems are very dependent on microelectronics; the special

needs of these systems must be recognized and designed into military microelectronics; and the expertise required for these tasks must be resident within the DOD and the AF (3; 16).

The Air Force Institute of Technology (AFIT) is a logical place to both introduce the military officer to microelectronic capabilities and design, and to prepare the officer for the design and implementation of microelectronics. A plan has been outlined at AFIT to conduct research into DA and to support the operational design needs within the Institute (30:1-2). The proposed DA capabilities needed at AFIT will attempt to:

1. Perform research in selected topics in design automation,
2. Develop an environment to support design automation research at AFIT, and
3. Develop a capability to assist digital system design and fabrication in-house at AFIT [30:4].

AFIT's future DA System design plans provide an excellent opportunity to design both the DA System and its IDDB. This project will provide numerous opportunities in research. These opportunities include the potential for designing and implementing the different levels of a DA system, data bases, application programs, algorithm development, artificial intelligence research, and other related areas of microelectronic design, development, and analysis. Probably the most important aspect of this development project is that the fruits of the labor can and will be

used to upgrade commercial, DOD, and educational facilities.

Many DA Systems have been developed from an existing design. This is because business considerations usually require that the system can partially be implemented at the beginning of the design. The design and implementation of a completely new DA System requires a lot of time and resources. The DA System being designed and developed at AFIT will be a completely new system. This report is a component of the overall AFIT DA System plan. The results of this report will provide a model of the DA System that is physically and logically independent of hardware and software concerns, with the designed-in flexibility to allow for the growth and changes that are characteristic of design systems.

CHAPTER III

DESIGN OVERVIEW

Techniques Used

Introduction

There are five major software engineering and data base design techniques that are used for the design requirements of this report. These techniques help to design and describe the IDDB, from the internal data requirements to the data activity in the DA System. The techniques used are:

1. Design Task Data Diagrams,
2. Design Cycle Activity Diagrams,
3. Third Normal Form,
4. Canonical Data Structures, and
5. Structured Walkthroughs.

The following sections of this chapter describe each technique. The final section discusses how these five techniques are used together to describe the design of the DA System and the IDDB models. The diagrams used, Design Task Data Diagrams and Design Cycle Activity Diagrams, are used similarly as the Data/Activity Diagrams are used by SofTech. However, in this report, the functions are reversed to stress the usage of the diagram. Thus, the data diagram is used to identify the data requirements

of a design task; and the activity diagram is used to describe the activities of the design tasks.

Design Task Data Diagrams

Design Task Data Diagrams consist of the function of the design task that the diagram is representing, the input data used, and the output data generated by the design task. The input consists of a task's actual input data and its control data. The input data is categorized and described by two generic terms: Input Design Data and Control Data. The outputs generated by the design task are:

1. Output Design Data,
2. Warning and Error Diagnostics, and
3. Execution Summary.

Figure 1 shows the input data on the left, control data above, output data on the right, and in the middle box the task being represented. The Data Diagrams show the specific and generic I/O data requirements for a design task, and these are expressed in two levels. These two levels can be seen in Figure 1. The first level contains only generic data descriptions, and the second shows the specific data descriptions. Only selected examples of the specific I/O data requirements of a design task are shown in the lower level Data Diagram. The specific data requirements are grouped into categories, the resultant

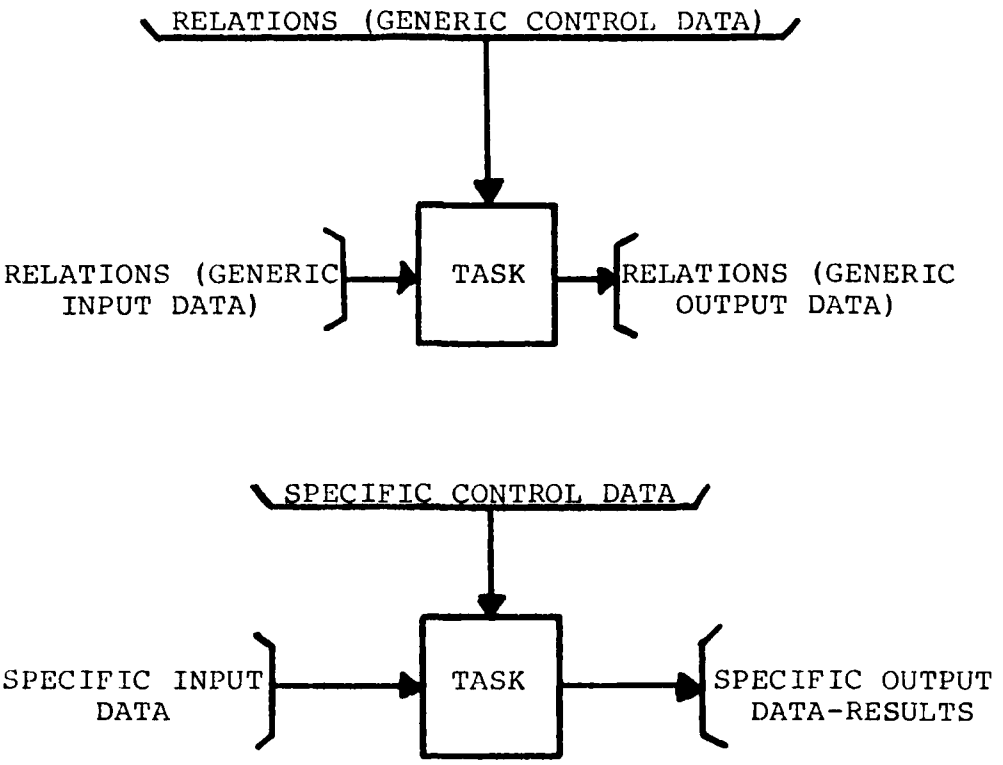


Figure 1. Examples of Design Task Data Diagrams

generic I/O data requirements are seen in this upper level of Design Task Data Diagrams.

Design Task Data Diagrams are also useful in understanding the conceptual schema. The conceptual schema is made up of relation schemes, and the GENERIC data that each relation scheme contains is denoted by the relation scheme's name. The conceptual schema defines and views the system's data organization at a high level. The SPECIFIC I/O DATA REQUIREMENTS, in the Data Diagrams, view the data at a low level. Conceptually, these two levels (GENERIC and SPECIFIC) define the relation scheme and the relation occurrences. It is also no coincidence that the GENERIC I/O DATA

REQUIREMENTS are also the names of the Relation Schemes that specify the design task's specific I/O data requirements. The relationship of the Data Diagrams to the data represented in the conceptual schema exists as a many-to-one mapping. The specific data requirements of many design tasks can be and are represented as a single occurrence of several relation schemes in the conceptual schema. This provides the mapping function (M:1), as illustrated in Figure 1 and Figure 2. Figure 2 shows how the entire IDDB is made up of the Specific I/O Data Requirements and that these are contained in the Generic Data Requirements.

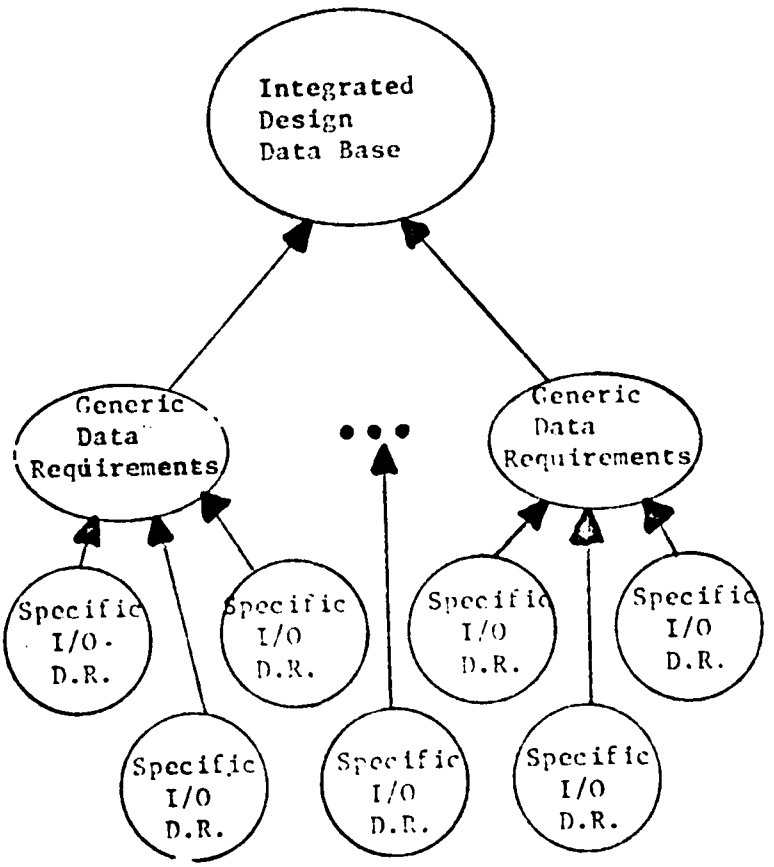


Figure 2. Data Hierarchy

Design Cycle Activity Diagrams

Design Cycle Activity Diagrams are similar in representation to Design Task Data Diagrams, except instead of showing how the data is transformed by an activity (design task), the Activity Diagram will show the flow of the different activities and their effects on the design data. Input, control, and output activities will be represented as arrows to/from the (design) data box.

The purpose of the Activity Diagrams is to graphically describe the changes that the design data goes through as the design evolves. The activities are represented by labeled arrows, and the box represents the effects that the activities have on the data. In other words, the arrowed activities transform the design data. Design tasks are grouped and represented as design phases. Each phase contains a set of design tasks that perform transactions on the design data. The Activity Diagrams shown are high level. Lower levels would be required for further implementation.

Activity Diagrams are useful in showing transactions on the design data by describing the design cycle. The diagrams show the data affected by design task activities. The Activity Diagrams will also show the point of return in the design cycle, to begin redesign, after modifications or corrections to the design have been made. Thus, in case of design errors the diagrams will show what

data has been changed by a task, so the erroneous data can be corrected.

Third Normal Form

This normalization technique prescribes ways to find the best grouping of the data, which will minimize the probability of future data base disruption. Without discussing the theory behind it (see Reference 25 for a complete discussion), the normalization process will be described. The following three steps will process data structures into Third Normal Form (3NF).

1. Decompose the data structures into two-dimensional tables, which describe relations between items.

2. Reduce the tables, so there is full functional dependence of the non-prime attributes on all the keys.

3. Eliminate the transitive dependencies of the non-prime attributes on all the keys.

The advantage of using 3NF is that the resulting relation schemes are now in a normalized form (3NF), that will resist common data integrity problems. These problems, which occur as a result of bad data base design, will be avoided through the use of 3NF. These data integrity problems include data base redundancy, potential inconsistencies (update anomalies), and insertion and deletion anomalies.

Another advantage of this form is that it allows a relation scheme to be decomposed and then joined, and the resulting scheme will be lossless. Also, functional dependencies of a relation scheme in 3NF are preserved after decompositions. These two characteristics (lossless decomposition and preservation of functional dependencies) are important because they provide proof that manipulations on the relation schemes will not lose information on data or data relationships. These proofs are important to guarantee integrity of the IDDB for each of the individual designers and for the overall data base (25-28).

Canonical Data Structure

A canonical data structure is used to represent the conceptual schema and the resulting representation is a canonical schema. The canonical schema is a normalized model of the I/O data requirements of the users (designers, tasks, DA System) of the design data. The importance of the normalization aspects of this model cannot be ignored. The normalization is obtained by requiring that all relations in the canonical schema be in Third Normal Form. This Third Normal Form normalization of the model precludes the possibility of the anomalies previously listed. This model of the data, canonical schema, represents the inherent structure of that data and also satisfies (by definition) the logical and physical independence goal of this thesis.

Canonical Schema. The particular format used to express the conceptual schema in this report is called a canonical data structure, and the result is a canonical schema. Usage of the terms, conceptual schema and canonical schema, will be consolidated into canonical schema. This is because a conceptual schema is a general representation of a canonical schema. The canonical schema will be used as a data organization and management reference tool to help in the implementation and maintenance of the data base. It will be designed and built using the canonical data structure technique which will be described next.

The following paragraphs describe developing, reading the graphics of, putting into perspective, and, finally, understanding, the canonical schema.

Development of the Canonical Schema. The next eight steps describe how to develop a canonical schema, which includes the 3NF normalization process (paraphrased from Reference 25:248-289).

1. Define a single task's view of the data, graphically showing the type of association (1 or M). The "task-views" in this case are the design task's view of the data.

No hidden transitive dependencies.

No redundant prime attributes.

(This essentially defines the Third Normal Form).

2. Define the next design task's view of the data, as above. Merge it into Step 1's result. Remove any synonyms and homonyms.

3. Denote the primary keys.

4. Add the inverse association between any keys. If an M:M association is created, remove it either by adding a concatenated key, or by dissolving one side of the association, if it will not occur.

5. Remove genuinely redundant associations.

6. Repeat steps 2 to 6 until all task views are merged.

7. Resolve isolated attributes and intersecting attributes.

8. Plan for future growth of the data base. Check if other relationships should be added, to prepare for future growth (new task-views). Remember, the canonical schema will easily accept data base growth, if properly planned for, so new tasks need not affect or, more importantly, invalidate the data base or the DA System design.

9. The canonical schema will be accurate so verify that all task views exist and are still in 3NF.

Definitions. The following terms define the symbols represented in Figure 3.

A Non-Prime Attribute is a data item or a piece of descriptive data. Non-Prime Attributes will be pointed at

Arrows: single arrow, unique directional mapping (1).

double arrow, multiple, directed, mapping (M).

For example:

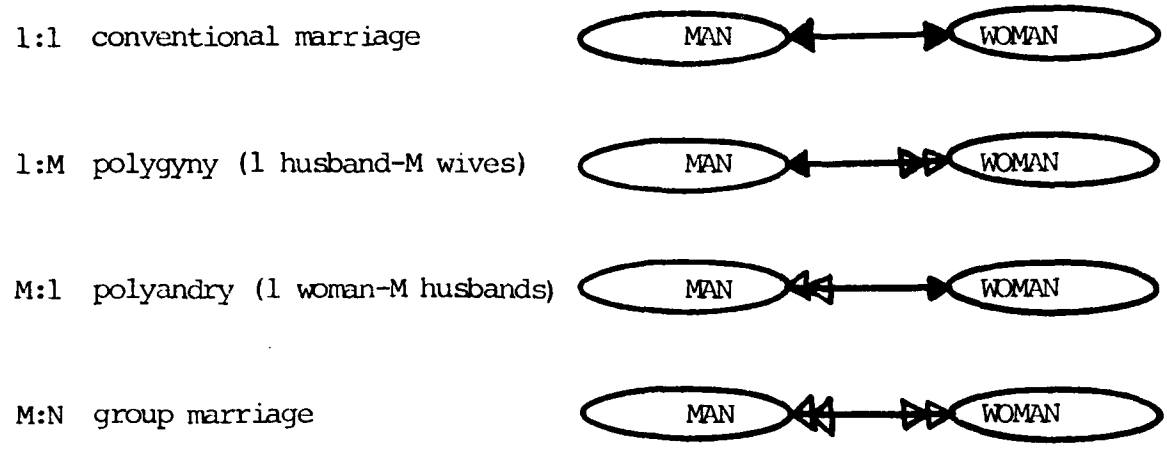


Figure 3. Canonical Schema Diagram
Symbol Explanation

by only one single arrow, its "owner," and Non-Prime Attributes will point at no other attributes.

A Prime Attribute is an attribute which is a member of a key, and cannot uniquely identify data (tuple). Prime attributes will be pointed at by single arrows, and will point with a double arrow. They can usually identify many occurrences of a data item, rather than unique occurrences. The prime attribute(s) constitutes a key and uniquely specifies data.

A key has two properties: unique identification and nonredundancy. The value of the key uniquely identifies the tuple. No attribute of the key can be discarded without

destroying the property of unique identification. Also, keys can uniquely identify their "owned" data attributes (details or assignments) and their prime attributes.

A Relation Scheme can represent an Association or "Details," which describe a relation or an occurrence. An "associative" relation scheme is made up only of prime attributes (called an association). A relation scheme that is made up of the key and its associated data attributes specifies a relation's Details. An example of an association in this report is the Signal-Pin Assignment, where each signal is associated with the pin of an electrical component. An example of a Detail would be the descriptive data for each component's pin, such as location on a board, pin function, impedence, etc.

It should be noted that a key that is subsequently used as a prime attribute will be shown graphically as a key function. An example of this occurrence would be when a Model is specified (i.e., Model Details), and then the data concerning the model's pins is required. The Model specification, which contained a key, is now a prime attribute (other prime attributes required for the key) because more information is needed to specify the Pin Details.

Putting the Canonical Schema Into Perspective. For future design and implementation of the IDDB, the next step would be to create the logical schemas and the physical schema of the data base.

Logical schemas are made up of the simple, task-specific, data associations that describe how the DA System users view their data requirements (task-views). Logical schemas are logically dependent on the AP. The specific data is input data, control information, and output data, along with the specific data formats required. Any information required for a specific AP (task view) must be provided in the logical schema.

The physical schema defines the physical location and characteristics of the data. It must take into account the access methods, read and write procedures, data formats, and other implementation considerations. The DBMS must be chosen and implemented before the physical schema is implemented.

Understanding the Canonical Schema. Each relation scheme must first be analyzed, and as the individual components of the design become familiar, then the entire design will be easier to comprehend. The associations that are used to describe the tasks' views are an invaluable aid in understanding the canonical schema. One task-view after another should be analyzed, to gradually become familiar with the data organization in the canonical schema. Once the design task's I/O data requirements (task-view) are understood (from the Design Task Data Diagrams) then the design phases (from the Design Cycle Activity Diagram)

should be analyzed. This will put the broader design objectives and data requirements into perspective. The Relation Schemes should also aid in describing both the data requirements and the data organization.

Some of the task-views in the DA System are represented in the Design Task Data Diagrams of Appendix A. The data requirements of several tasks are shown in these diagrams. These data requirements, as listed in the Data Diagrams, are defined via the Relation Schemes and the Details Sections which describe the canonical schema. The data items, which comprise the non-prime attributes may be contained within a group of other related data items, which will make up a section of data called Details.

Structured Walkthroughs

Structured Walkthroughs were used to verbally describe various aspects of this report to an audience consisting of potential users of its results. These presentations were used to ensure that the potential users understood the thesis design, its interim results, and that the design work being done was correct. It provided an interactive, dynamic forum for the design of the IDDB and the DA System. As a result of these meetings, many changes were made to the design.

Technique Interactions

This section will describe how the five previous software engineering and data base design techniques have been used to design and verify the canonical schema. As previously mentioned, the structured walkthroughs were used to validate the design results. The Data Diagrams are expressed at two data levels: specific and generic. The generic data names used in the Data Diagrams are also the names of relational schemes used in the canonical schema description. There are many occurrences of the tasks's generic data (in the Data Diagram) which maps onto a single relation scheme; this is a many-to-one mapping). Many tasks may require the same generic data, which is specified through one relation scheme. The canonical schema is a complex structure, but is constructed of relation schemes which are in 3NF. The Data Diagrams show how the data fits into the canonical schema, making it understandable.

The Activity Diagrams show, at a high level, how the data is affected by the different tasks in the design cycle. These diagrams will become more important, and will have to be done at a lower level, during further implementation of the design of the DA System and its IDDB.

Data Abstraction Hierarchy. Design projects describe inherently complex systems, with a tremendous variety of information. Because of this complexity, there

is a need to conceptually and hierarchically organize the design data, which makes up the IDDB, is required by the DA System, and will be used in the design cycle. A Data Abstraction Hierarchy, Figure 4, consists of the data that is contained in the IDDB and which redundantly shows this data at different levels of detail and in different forms of representation (1:1259).

The Data Abstraction Hierarchy is shown in Figure 4. It shows the IDDB decomposed. The diagram shows how the data requirements of the Design Cycle, starting at the bottom, are redundantly specified as Specific I/O Data Requirements (Specific I/O D.R.) of the many APs that are contained in the Design Cycle. Each of the Specific I/O Data Requirements is logically dependent on the APs. The data of several APs may be the same and is grouped into Generic Data Requirements (Generic D.R.). Thus several specific I/O Data Requirements may be represented by a single Generic Data Requirement. Figure 4 shows that data abstraction occurs between the Generic and the Specific I/O Data Requirements. It also shows that the entire data requirements of the Design Cycle are contained in the IDDB.

Design Approach

Introduction

The design approach taken during this report is composed of four steps:

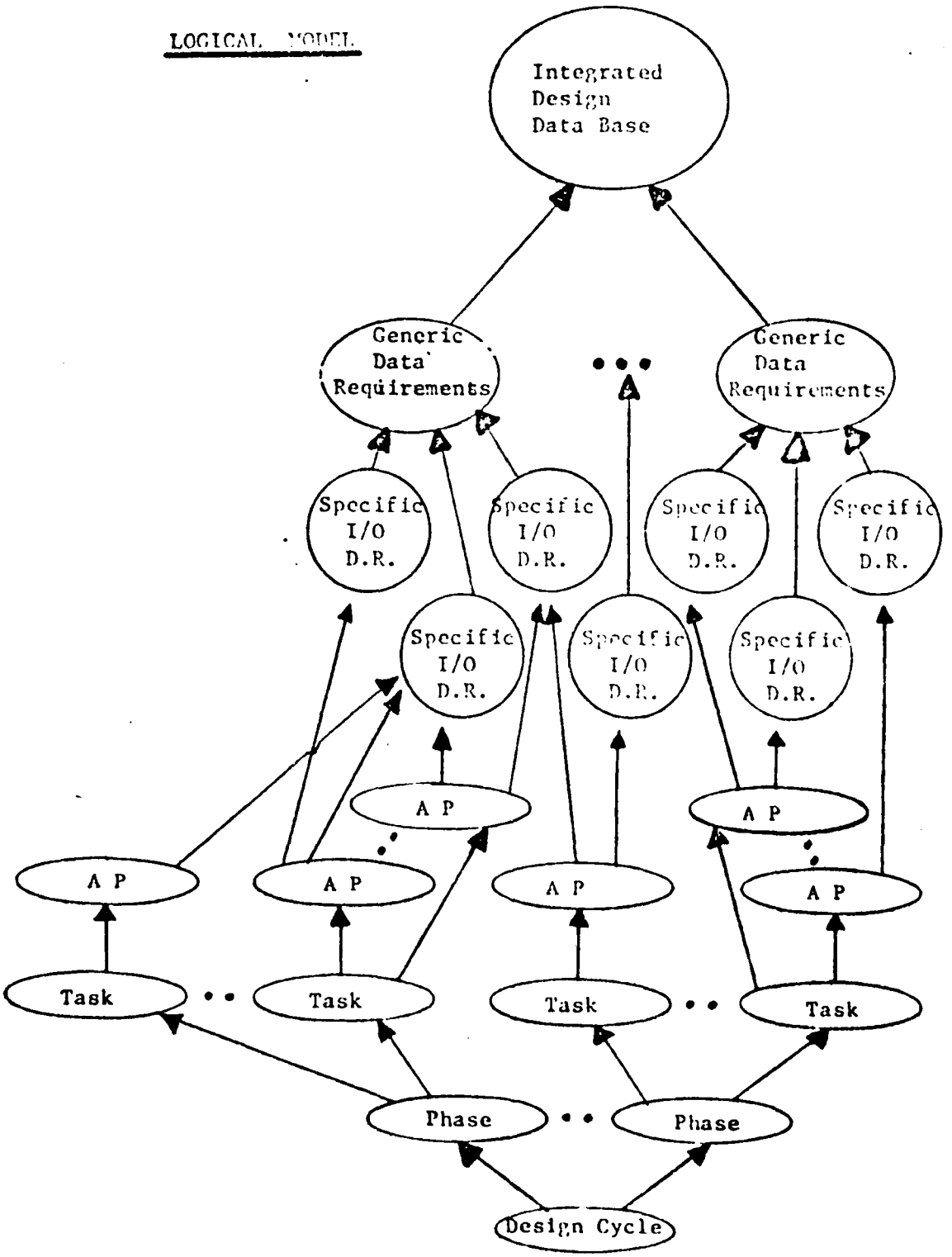


Figure 4. Data Abstraction Hierarchy

Step 1--collect the user's design requirements;

Step 2--collect the total system requirements;

Step 3--build the canonical schema; and

Step 4--document the data organization, in terms of the requirements for both the users' and the system's views.

Step 1--Collect User's Design Requirements. This first step of the design approach is made up of two sections: (1) identification of the task-views, and (2) characterization of the data required for each task-view. A questionnaire was used to gather the information required to characterize the data. Many of the results of Step 1 are presented in Appendix A which contains the Design Task Data Diagrams.

1. Identification of the Task Views. First, the different design tasks to be used in the DA System must be identified. (Later additional tasks may be added, but an initial group of tasks must be used for the specification requirements of the D/A System.) As soon as a task has been identified, the Input/Output Data Requirements (I/O DR) must be listed and defined. All of the task requirements together define the data requirements of the DA System.

2. Characterization of the Data Required for Each Task-View. Generally, the questionnaire was used to

identify the data required for the design of the IDDB and the DA System. Specifically, the purpose of the questionnaire (see Appendix C) is to ascertain various facts about the data items to be used in a task and the associations that exist among the data items. The data item's name, definition, attributes, known dependencies, and unique identifiers are helpful in characterization. Associations between two data items require an association name, the data items involved, the mapping property (1:1, 1:M, or M:M), the implication or meaning of the association, and a list of any unused but meaningful associations. The final results of an answered questionnaire will be a "task-view" of each design task.

The questionnaire was originally intended to be answered by microelectronic designers; however, the technical level of the questionnaire required too much knowledge of data base design. Also, the short time frame of the thesis report prevented the questionnaire's use as intended. Instead, the author used the questionnaire as a guideline to follow in researching the many design tasks that designers need to have performed.

Step 2--Collect System Requirements. This second step in the design approach is made up of two sections: (1) definition of the data base environment, and (2) characterization of the interactions of the design tasks. This

step satisfies the solution to the second goal, as stated in Chapter I: Design the DA System, and use the design as a model environment for the IDDB.

1. Definition of the Data Base Environment. The data base environment, as defined in this paper, is the DA System that will generate and use the data that is in the IDDB. The approach taken in defining the environment is to break the DA System into its component parts, describe each part, describe the interactions between the parts of the DA System and, finally, show how the IDDB will be organized and used in this environment. This definition of the environment will be found in the DA System description.

2. Characterization of the Interactions of the Design Tasks. To characterize the interactions of the various design tasks to be used in a DA System requires describing the precedence ordering of the different tasks, including tasks that can be performed independently or concurrently, the data changed per task, the restart point for error recovery, and the control loops. Two levels of this resultant characterization can be seen in the Design Cycle Activity Diagrams.

The results of Step 2 are presented in Chapter IV. The data base environment is described as in the DA System, its organization and use. The interactions of the design tasks are described in the Design Cycle Activity Diagrams.

Step 3-- Build the Canonical Schema. This third step in the design approach entails integrating and minimizing the results of the previous two steps (Collection of User and System Requirements). A canonical schema is produced from this effort. The canonical schema is a model that represents the entire information content and organization of the Integrated Design Data Base. As previously stated, the design of a canonical schema implies that there will also be logical schemas (made up of each design task's view, specific data requirements), and a physical schema (the physical implementation design of the data base). Chapter V describes the design process and the resultant model of the IDDB.

Step 4--Document the Data Organization. The data organization must be documented for both the task and the system in terms of the canonical schema requirements. Some of the task-views are described in the Data Diagrams in Chapter IV. The specific input and output data requirements of these views are mapped into generic data categories and then respecified into relational terms as seen in definitions of the Relation Schemes and the Detail descriptions of Chapter V. The data base environment is described in the DA System sections, Chapter IV, and the interactions of the design tasks are found in the Design Cycle Activity Diagrams of Chapter IV. Finally, the canonical schema is completely documented in Chapter V.

CHAPTER IV

DESIGN OF THE DA SYSTEM MODEL

Introduction

This chapter is divided into three sections. The first section discusses the general assumptions concerning the DA System, the second discusses the organization of the DA System (which contains the IDDB), and the third discusses how the DA System will be used. The individual and grouped assumptions that follow should help clarify concepts affecting the DA System design. (Note: the DA System is represented in Figure 5 and the reader will find it a useful reference during its discussion.)

General Assumptions

Definition of Terms

While reading this thesis, there may be some confusion as a result of term usage. It has not been possible in this report to use normal terms from only one area of study because the report's subject is interdisciplinary. The author has attempted to make the meanings of words clear and to use words consistently in the generally accepted sense of their meaning. To help, a Glossary of Terms (Appendix B) has been provided. It is suggested that

the glossary be skimmed to verify and recognize the different definitions before reading on.

Design Model Representation. A design may be implemented using many technologies and models. Physical implementation of the design may be postponed for quite awhile during the design cycle, because specific technology designation may not be immediately required during the logical design tasks.

Project ID. Each design project is assigned an identification number, ID#, which uniquely identifies a particular design effort. If the effort is broken into teams, then a multi-value identifier may be used. The identifiers used may be: Team # and Designer #. This will prevent data integrity problems such as when several designers working on the same design are independently designing the same segment of the design. The important point is that the identifiers can be lumped under one identifier key: ID#.

Project Independent and Project Dependent Data. Any data items that can be assessed without an ID# as a part of its key is Project Independent (PI) data. Project Independent data is data that is used for references by the designer and is not changed by design work. PI data values are independent of any particular or individual project.

PI data is used for reference only, and values of each PI data item can only be changed by the DBA, not the user.

Project Dependent data is data that is generated specifically for a particular design and is changed by design work. Any data items that must be accessed with an ID# in the key is Project Dependent (PD) data. PD data values are dependent on a particular, individual project. PD data can be dynamically changed during design tasks. The designer can change an incorrect part of the design or the AP can change the PD data during the design task. All data items are either PI or PD data. Once classified as either PI or PD, this classification is fixed, although the DBA can make PD data into PI data stored data.

The design tasks in this report, the APs and their associated data requirements, are not defined or described exhaustively. However, these design tasks and data requirements are representative of the techniques and design data that designers are using to design microelectronic systems today.

DA System Organization

A conceptual view of the DA System's organization shows the three separate data areas of the IDDB which consist of:

1. Project Independent (reference data),
2. Project Dependent (designer's design data), and

3. Execution (task Data).

The DA System which contains and uses the IDDB is also made up of:

4. A Data Dictionary,
5. An Executive,
6. Application Programs,
7. Application Program Interface,
8. A DBMS,
9. A DBMS Interface, and
10. The Users (called designers) of the DA System.

This DA System organization can be seen in Figure 5, and will be helpful for the following discussions. By definition, the data is an integral part of the DA System. The following subsections discuss each of the above elements of the DA System.

Data Areas of DA System Model

The IDDB permanently stores two logically different data bases. These are stored as the Project Independent (PI) and the Project Dependent (PD) data areas. Each designer has his/her own PD data area. During actual design work, using the DA System, a third data area is created, which is the Execution data area. This is where the design data required for a task, from both the PI and PD data areas, is generated for use during a design task execution. According to some authors, these three data areas could

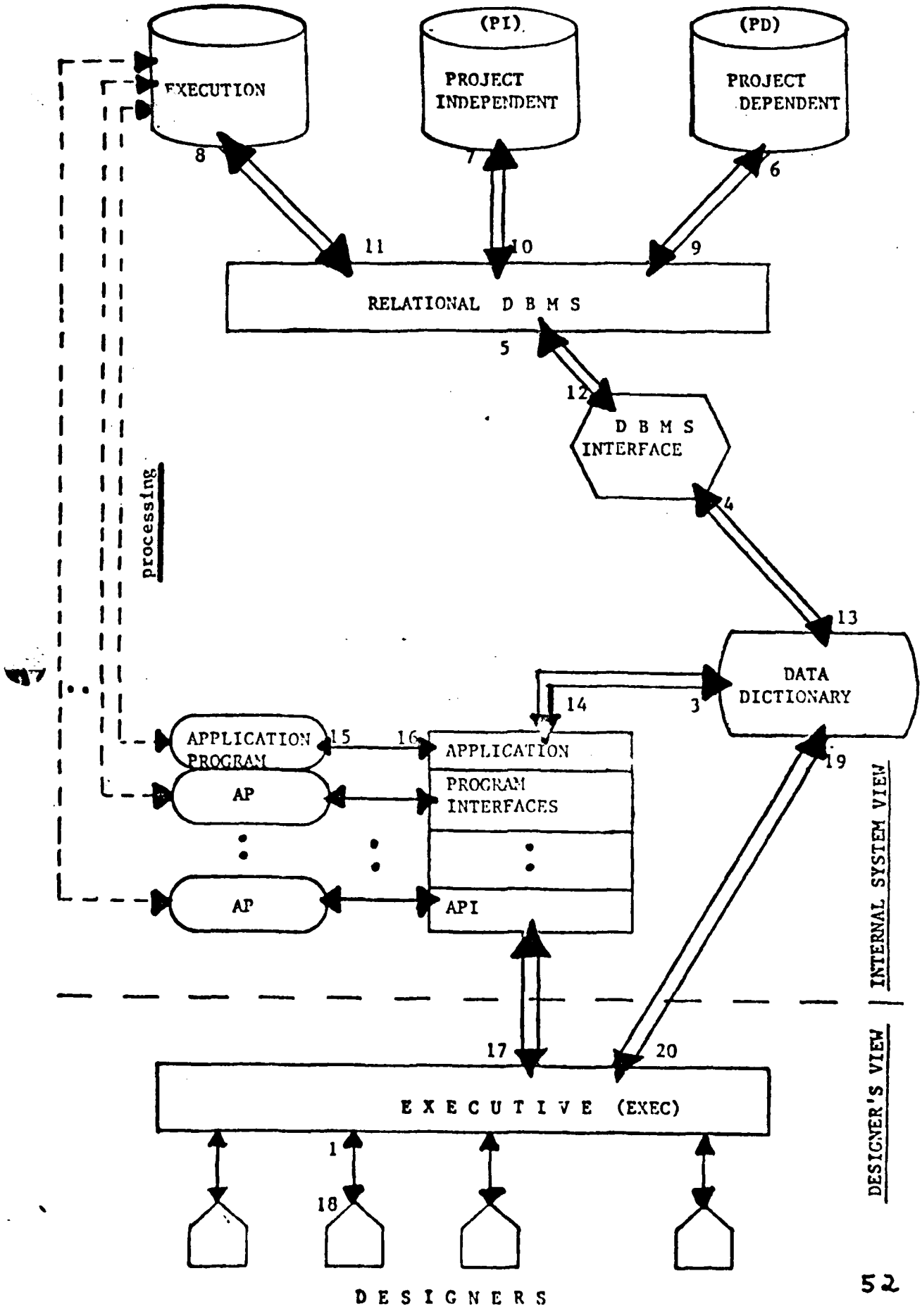


Figure 5. DA System Conceptual-Level Model

also be called "work files," where

> . . . portions of the main data base are extracted to form work files, which can then be customized by a set of applications to provide efficiency and to eliminate data contention problems at the central data files [7:94].

These three data areas are now described.

1. Project Independent (PI) Data Area. The PI data area is made up of reference data. This is data that the designer and the design tasks reference and use, but cannot change. Examples of this are Model Details, Defaults, and Macro Nets. The DBA controls the maintenance of this data area.

2. Project Dependent (PD) Data Area. The PD data area holds each designer's design data. This is data that is defined/changed dynamically during design tasks or interactively by the designer. Examples of this data are the Design Net, Default Changes, Design Details, and logical and physical designs. The designer controls the maintenance of this data.

3. Execution Data Area. The Execution data area is the scratch data area into which the PI and the PD data are written at execution time for use by a specific task. Each task views data as it is defined in the canonical schema; therefore, the data sources, PI and PD, are irrelevant. The data that will exist in this Execution data area during task execution is the data specified by the Input Data Requirements Relation (to be discussed),

and any internal data created during the design task's execution. After inspecting the resultant data, the designer will have the option to save it in a PD data area or delete it from the Execution data area.

Functional Components of
DA System Model

The following components of the DA System are the "functioning" components and provide services to the designers and the design data.

4. Data Dictionary (DD). The Data Dictionary contains the definitions of all the data items, where they are used, data types, etc. The DD also contains the logical location of each data item. The logical location essentially defines the DBMS query to generate the relation that a data item belongs to. The Data Dictionary is an important DBA tool, also.

4. Executive (EXEC). The DA System's Executive is the software interface that the designer interacts with. The designer tells the EXEC what design task is to be executed and the EXEC performs those tasks. Any pertinent questions, data error checks, or other person-machine interaction is coordinated by the EXEC. For example, when a designer's design data is insufficient for a particular task to execute, the EXEC is notified by that task and the designer is requested to provide/correct the data. Note

the design data contains the stored data (previously defined or generated), and the input data (designer input for this task). This is an important characteristic, since the design data that was correctly generated by a previous design task and verified by the designer is the same design data used in the next design task.

The data is not stored redundantly in the different data format required by the different AP, but rather is stored in a single implementation of the canonical schema. Also, the next design task used will need only small amounts of additional data for its input and control data requirements.

The EXEC knows which APs make up the functional requirements of a particular design task. The EXEC will provide the designer access to the PD data area, which allows the design data to be viewed or changed. The PI data can also be reviewed, and the default values of a design task should be checked by the designer through the use of the EXEC.

The EXEC should manage the design data thus demanding less of the designer's time. These functions will lessen the burden on the designer by providing a single, flexible, easy-to-learn interface to the DA System. Thus, the EXEC will allow the designer more time to concentrate on the formidable tasks: to design better microelectronics in less time.

6. Application Programs (AP). There are many Applications Programs that make up design tasks that are used by a DA System, and new ones are constantly being replaced and added to the system. These APs are the software packages that perform the calculations and manipulations of the design data. For instance, an AP may be a component placement program for gate array layout. An AP may make up an entire design task, a part of a task, or an AP may contain several tasks. An important point is that each task is defined to be one task-view as far as the logical schema is concerned. Thus, several APs, using different data formats and data item names, may have to be executed to perform one task.

7. Application Program Interface (API). Each AP has an Application Program Interface which is made of a subset of two Relation Schemes: Input Data Requirements (input data required to run the AP) and Task Results (output data from the AP). (These relation schemes are completely defined in Chapter V.) The APIs, which represent the APs used to make up a task, provide the relation scheme subsets that are to be JOINed, and the result is the complete relation scheme. For example, the design task may include placement and route functions. The Input Data Requirements of both of the APs will be specified by the APIs which will allow the needed data for the design task to be generated for use by the APs.

An API also stores the actual format of the data required by both an AP and the Data Dictionary. As implied by the previous discussion, the complete relation scheme defines the upper-level description of the logical schema. Thus, when the relation scheme subsets are joined, the resulting complete relation scheme contains the logical scheme. The Data Dictionary transforms the data provided by the DBMS, and gives it to the AP, and when data is generated that is to be stored by the DBMS, the Data Dictionary transforms it and gives it to the DBMS (via the DBMS Interface). This interface function should be transparent to the designer.

8. Data Base Management System (DBMS). The DBMS is the software that must handle all access to the IDDB. This single interface to the IDDB provides for consistent data operations, reduced data redundancy, shared data, secure data, and enforced standards. There must also be control mechanisms to protect users of the data from compromised data (data integrity).

9. DBMS Interface. The DBMS Interface receives the logical locations of the data and transforms them into DBMS queries. The DBMS Interface transfers the queries and query results to the appropriate DA system component. The frequently-used DBMS queries can be optimized, and this would be the location of the query optimization routine.

10. Designers. In a DA System, the designer is the most important component. It is the designer's expertise that the design system must be capable also of incorporating into the design. It is the designer's time and the amount of data manipulations to be performed that must be minimized (24:286).

DA System Use

The description of the usage of the DA System will be in two parts: the designer's point of view and the internal interactions of the DA System's software.

Designer's Point of View

Basically, the designer will only interact with the EXEC. When a designer first logs into the DA System an initialization procedure is executed which identifies the designer and the design tasks to be run. Any extra data that must be provided by the designer will be prompted by the EXEC. Questions concerning any design tasks, data required by a task, and even questions concerning the present design are all handled by the EXEC. The EXEC will facilitate changes to the design data, and will coordinate changes to program default values. In summary, the designer will only interact with the EXEC and the EXEC will remove or reduce many mundane design tasks.

The function of the EXEC and the resultant view of the DA System that the designer sees provides a simple, but

flexible interface for the designer. While the design of the EXEC is original, similar concepts have been used in other systems such as the Computer Aided Design and Design Automation System (CADDAS) at the Advanced Technology Laboratories, RCA Corp. Reportedly,

> . . . [they] designed a system which executes application software through the use of control executives. These executives automatically gather the appropriate data, perform any necessary manipulations, and cause the execution of the application program [24:285].

Internal Interactions

These interactions are described in three phases: (1) Execution Data Generation, or preprocessing; (2) Task Execution, or processing; and (3) Task Completion, or postprocessing. (Note: the numbers in parentheses will indicate their location on the DA System diagram (Figure 5).)

1. Execution Data Generation. At the beginning of a design session, when the designer requests the EXEC to run a design task, the data required to run this task must be gathered into the Execution data area. The EXEC determines the series of actions to accomplish the designer's request. The EXEC calls (2) the APIs that are required to run the APs that make up a design task. The partial Input Data Requirements (IDR), specified in the APIs, for each AP are JOINed by the Data Dictionary (DD) to form the full Input Data Requirements (IDR) Relation. The IDR specifies all of the design data required by the

task. (Further discussion on the IDR relation is given in Chapter V.) The DD then specifies (4) the logical locations of the data to the DBMS Interface. The DBMS Interface converts the relation's logical specification into DBMS commands (relational queries) that will be submitted to (5) and executed by the DBMS.

2. Task Execution. Before task execution is begun there are default parameters to be verified and approved or changed by the designer. The EXEC coordinates (17) the default specifications of each task. The designer is provided (18) with the task Defaults for inspection, so that any changes (Default Changes) can then be interactively made. The Defaults to be "Changed" are flagged (7) during the interactive session because the Defaults are PI data, and then the Default Changes are stored (6) in the PD data area. These Default Changes cause changes (8) to be made in the Execution data area also. Any design data (9) that has not yet been provided by the designer, for use by the task, will be requested (18) from the designer by the EXEC and will be added to the designer's design data. Once all the design data has been completely loaded and verified, (9, 10, 11) task execution can begin.

3. Task Completion. During the task execution, Task Results data is generated. When the task execution is complete the new design data must be evaluated. This data, called Task Results, will be placed in the PD data

area, but only after the results have been checked and approved by the designer.

Often the resultant data (11) (Task Results), will be error messages and execution diagnostics. The designer will need to reference them to help correct the design data. While these messages may not be important design data, they may be critical to evaluation, redesign, or data correction activity required by the designer. Once the corrections have been made to the design data the error messages can be discarded or saved.

For the resultant data to be saved the designer must approve the Task Results data for storage. When approved for storage the API will notify (3) the DD, which in turn will provide (4) the logical storage location for the data to the DBMS Interface. The DBMS Interface issues the retrieval (5) (from the Execution data area), and the storage (to the PD data area), commands (8) to the DBMS. The DBMS then proceeds to execute the commands to complete the storage operation.

DA System Model

The following is a more detailed description of the DA System functions at each critical point (numbered arrows in Figure 5) in the DA System.

1. The designer provides inputs, commands, questions, and responses to the DA System, via the EXEC.

2. The EXEC selects the APs to execute, which together constitute a design task. This is also where the EXEC responds with the additional data from the designer required by an AP to execute or with the disposition commands concerning the Task Result data.

3. The partial IDRs Relations are joined in the DD to specify the full data requirements of the design task. The IDRs request information from the DD that is needed by the APs to execute.

4. The DBMS Interface receives the logical location from the DD of the data needed (to read or write) and converts it into the DBMS commands required to process the data.

5. The DBMS receives the commands from the DBMS Interface and executes the commands against the PD, the PI, and the Execution data areas.

6, 7, 8. These are read and write commands that the DBMS executes against the three data areas.

9, 10, 11. The results of the previous DBMS commands are generated by DBMS command executions. A result can be a successful read (thereby transmitting the data), a write (changing or adding data), or an unsuccessful read. In the case of an unsuccessful read the EXEC is notified (12), (13), then (20). The EXEC determines what data must be added to correct the situation. This decision is the result of a query against the APIs to

find the needed data. If additional PI data is needed, the DBA must be notified. That would mean that the required system data is missing. If additional data is needed for the PD data area, the designer is requested (18) to provide the appropriate design data.

12. This is the receiving side of the pipeline that exists between the DMBS and the DBMS Interface. All the information that is requested by the DBMS commands (5) is funneled back through the DBMS Interface (12).

14, 20. The requested data is passed to the DA System component that originally requested it (14 or 20). If the data is to be used by the APs, it will be passed to the APIs by the DD and then passed to each respective API (14). Each API will in turn provide the needed information to its associated AP (15). If the information was requested by the EXEC (see #19), then it will be sent to the EXEC (20).

16. As a task executes, the resultant data, Task Results, will be transferred (16, 3, 4, 5) to the Execution area (8) as the data is generated. (This may not be efficient, but it has no detrimental effects on the DA System as a data base environment.)

17. The EXEC is notified when the proper data is not available for a design task execution. The EXEC is also queried as to the disposition of the Task Results after a task's execution.

18. The designer receives the queries and responses from the EXEC.

19. There are circumstances in which the EXEC must query the data base. These include times the designer wants to see parts of the design data, PD or PI data area. The designer may also wish to change parts of the PD data when Default Changes are made to the PD design data.

Design Cycle

Introduction

The Design Cycle presented here uses general terms, called generic descriptions, to describe the different design tasks. The Design Cycle or Design Cycle Activity Diagram, presentation also represents the transformation of the design data during the Design Cycle. These design tasks in the Design Cycle are grouped into six phases which make up the design cycle:

1. Input Design Specification,
2. Logic Design,
3. Logic Design Verification,
4. Physical Design,
5. Physical design Verification, and
6. Final Artwork and Documentation.

Design Cycle Phases

The following descriptions will discuss these six phases as they are represented in Figure 6.

1. Input Design Specification Phase. This phase includes both logical and physical designs. Initially, the design is logical in nature. The design will gradually evolve into a physical representation during the course of the design cycle. The techniques used to specify these design representations are varied. They can vary from graphical (using interactive graphics systems), logic design languages, hardware description languages, stick diagrams, block diagrams, functional specifications, register transfer languages, and behavioral specifications. Design languages, in general, are either structural (define logic elements and their interconnections), or behavioral (define how each element functions logically and timewise).

Usually the physical design specifications use the verified logical design and transform the logical design into the physical design. But before the physical transformation is done, the logic design must be verified. Also, because the logical design is general or non-specified, it can be physically implemented using many technologies and electronic "parts," ("silicon," DIP components, hybrid circuits).

2. Logic Design Phase. Once the initial logic design has been specified, the Logic Design Phase consists

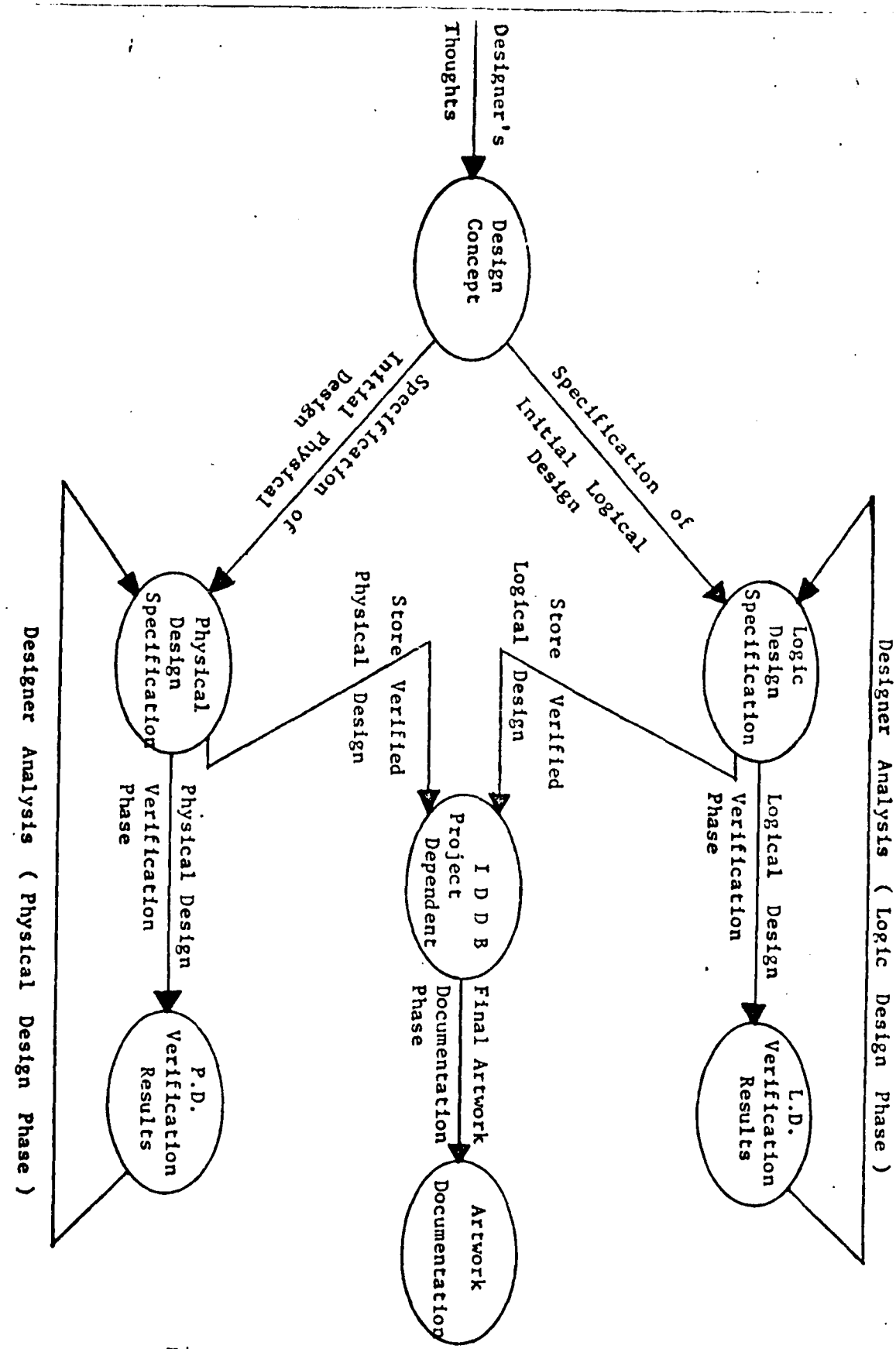


Figure 6. Design Cycle Activity Diagram

of the analysis performed by the designer on the results of the Logic Design Verification Phase. There is a cycle between the two phases that continues until the logical design is verified. The Logical Design Phase is performed by analyzing the results of the Logical Design Verification Phase. The cycle among the two Phases entails iterations of simulations and other analyses, which make up the Verification Phase. Then the designer analyzes the results of the verification tasks, changes the design; this is the Logical Design Phase. Then the Logical Design Verification Phase can begin another cycle.

3. Logical Design Verification Phase. This phase includes Logic and Fault Simulations, Controllability/ Observability (C/O) tests, and Timing Verification tests. Logic design verification allows the designer to verify that the computer representation of the design will logically function precisely as envisioned by the designer. These different verification tasks exercise the design, attempting to find certain design problem areas such as signal faults, timing errors, and internal C/O weaknesses.

4. Physical Design Phase. This phase includes those tasks that transform the present design into a physical implementation of the logical design. Often these design tasks are iterated with the next Phase, Physical Design Verification. This iteration within the design cycle allows the physical design problem areas to

be flagged before the design goes too far. The design may also be segmented, and then designed and verified in sections.

Because a logical design can be physically implemented using different technologies, approaches (IC, PCB, Gate Array, etc.), design rules, and geometries, it is important for a designer to be able to explore the effects of many implementations of the design. Thus, the designer's goals for the design can critically affect the course of this Phase of the Design Cycle. For instance, optimizing for hardware or software speed, physical size, reliability, maintainability, cost, testability, or a combination of several of these factors, affects the design cycle and the results of the design tasks. These factors also influence the actual design tasks to be used within the design cycle.

5. Physical Design Verification Phase. This phase includes tasks such as design rule checks, electrical rule checks, and specification rule checks. As previously noted, these tasks often interact in a cyclic manner with the Physical Design Phase. This phase can be used to partially or completely verify the physical design early in the Design Cycle. The rule checks entail physical geometric checks, electrical constraint tests, and verification on the logical and physical designs' equivalence.

A. R. Newton (12:1189), provides support for these phases when he discusses different techniques used for

verification. These techniques are factored further into functional and physical aspects of the design cycle. The logical symmetry of the design cycle provides the symmetry seen in this upper level Design Cycle Activity Diagram.

It should be noted that the tasks within the Design Cycle define a structure of data dependencies. These data dependencies exist as a result of precedence ordering among the design tasks that generate or change the data. To ensure data consistency and design integrity, these data dependencies must be kept valid. Some design systems only allow a fixed design cycle, with the dependencies defined only in the forward direction which prevents data dependency problems. However, designers find this restrictive. A design system with a fixed or a "disciplined design methodology" is described in Reference 5. This system achieved all its design goals. However, the designers were not consulted concerning the fixed design process, and consequently rejected the system (5).

The DA System model designed in this report allows iterative loops, which are at the designer's discretion. Design Cycle Activity Diagrams, at different levels, will provide useful documentation showing affected data for each design iteration.

6. Final Artwork and Documentation Phase. Once the design has been completely satisfied, Phases 1-5

completed, the artwork required to physically fabricate the design must be generated.

Extended Data Abstraction Hierarchy

The Extended Data Abstraction Hierarchy (Figure 7) shows the conceptual and logical models which present two representations of the IDDB's data. This previously discussed data hierarchy shows the way that the different APs, Tasks, Phases, and the Design Cycle view their data requirements. Note that an AP requires several Specific I/O Data Requirements to provide the data needed. Figure 7 also shows how descriptions of the models and their components are related. These components include the Canonical Schema (Figure 8), the Relation Schemes and their definitions, prime and non-prime attributes, the two levels of data requirements (Generic and Specific) of the Design Task Data Diagrams, the Design Cycle Activity Diagrams and its six Phases. The canonical schema and the IDDB are pictured as parallel representations with the dashed line between the two signifying the shared names of the Generic Data Requirements and the Relation Schemes. Figure 7 also shows the relationship of the keys, relation details (non-prime attributes), and the definition of the relation schemes.

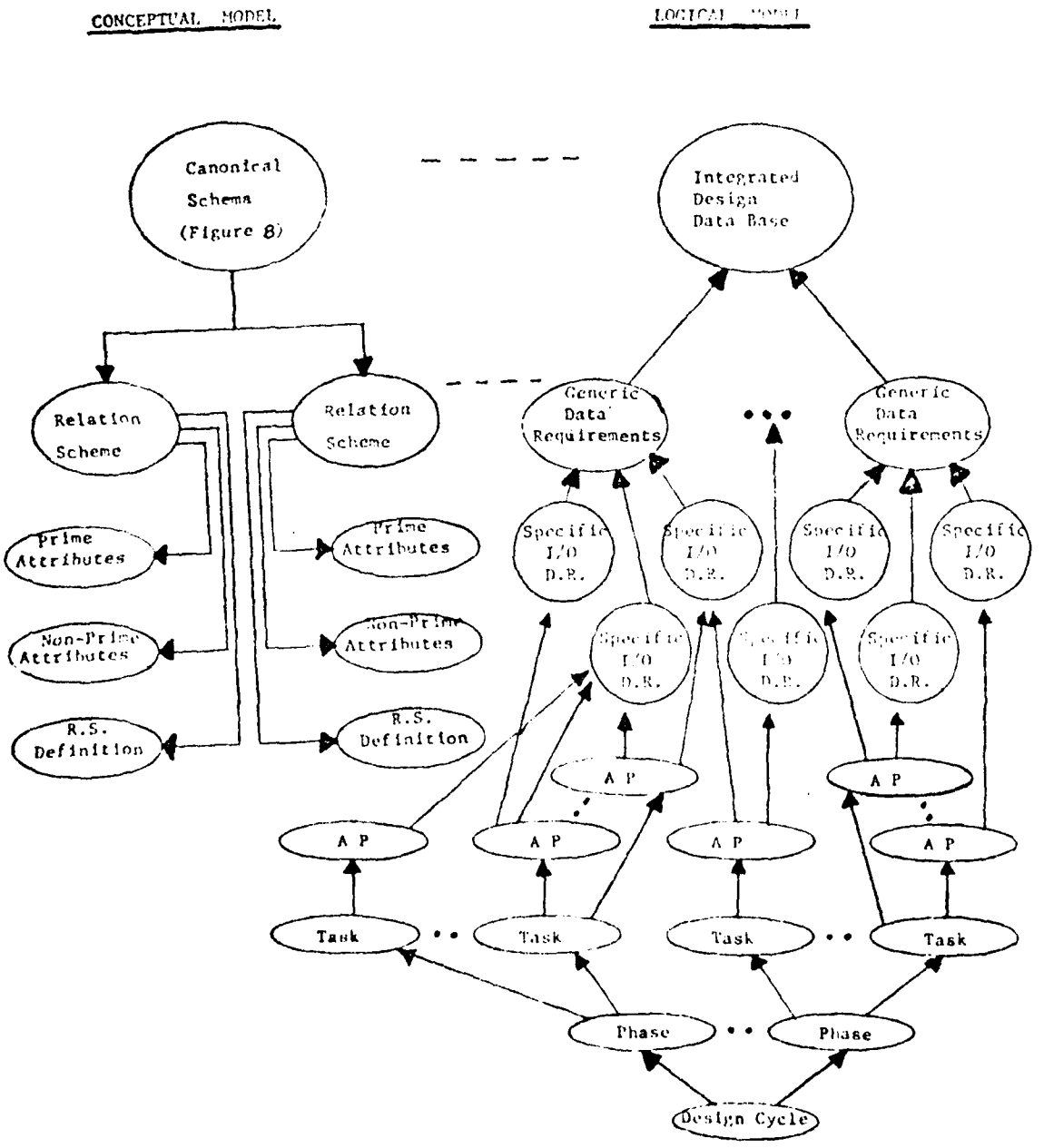


Figure 7. Extended Data Abstraction Hierarchy

CHAPTER V

DESIGN OF THE CANONICAL SCHEMA

Introduction

This chapter describes the design of the canonical schema of the IDDB. The chapter is broken into three parts: Assumptions, Elements of the Canonical Schema Design, and Meaning of the Canonical Schema.

General Assumptions

The following section describes assumptions made prior to the design of the Canonical Schema.

1. Data Item Types: Data items are categorized into groups of "types." The data items are grouped by the way of Application Programs (design tasks) use and data items. Thus a data item may be used in several "type" categories. For example, signals to be used in a design would be a data item. Each Signal is named in the design and has signal values. Depending on how the Signal is used, will specify how the different Signals are grouped into "Types." There may be several "types" of Signals:

a. Fault (signals to be faulted),

b. Sample (signals to be sampled and used in a timing diagram),

c. Undefined (signal values presently undefined), and

d. Case Analysis (signals to be used in several types of analyses).

This "Type" concept is expanded in the next set of assumtions.

2. Model Specification. During the physical design phase, the technology and the models used for Cell Pattern (CP), Component (COMP), Macro must be specified. In other words, a logical design may be physically implemented in many ways and, as such, additional data is required to specify the physical implementation of a design. A Model specification will require, as explained below, the Element Type (COMP, CP, Macro), Model Type, and Technology.

There are many possible representations of a COMP, CP, or a Macro. These different representations are classified as Model Types. Model Types are the different forms, in a logical sense, that a COMP, CP, Macro can assume. For example, a COMP #, 7400 (maybe a NAND gate package) may have many representations (Model Types). These Model Types may include a physical outline, an internal schematic, internal physical descriptions, a C/O model, a Timing model, a Cost/Reliability model, a Heat Dissipation model, or any other model.

A COMP, CP, or Macro can also be implemented using different technologies. This is the third and final prime

attribute (the first two are Element Type and Model Type) required to uniquely identify the model to be used. A Model Type has a different representation for each Technology. Thus, a NAND gate COMP may be represented by ECL, TTL, CMOS, or some other technology. While some Model Types of different Technologies may be the same (such as the COMP Outline of a package), proper data base design techniques (3NF) demand that these (presently) identical models be kept separate.

3. COMP and Macro. No attempt will be made to rigorously define the distinction between CPs and Macros. The assumed definition states that Macros are made up of CPs, and that, logically, Macros are high level while CPs are low level.

The value ranges that COMP#, CP#, Macro# can assume are mutually exclusive. A CP# uniquely specifies a CP, and no COMP or Macro will have the same identifying number. Relations #7, #8, #9 (Element to Macro, CP, COMP Assignments, respectively, to be described later in this chapter), may seem redundant because they are mutually exclusive. However, it is important to keep their individual designations to aid the designer's and DBA's understanding of a design specification.

4. Relation Schemes and Attributes. Relation Schemes either (1) define an association among data items (prime attributes), or (2) define a data item and

its "Details" (non-prime attributes). The Details category consists of the non-prime attributes that describe a specific relation, and the descriptive data is only relevant for a uniquely defined occurrence of a relation scheme or a relation. For example, Details can include descriptions of the following types of design information:

a. physical (# of pins, # of gates/pack, design rules, pin type, X/Y offsets of pins, etc.);

b. electrical (power rating, resistance, tolerance, load current of a signal pin, etc.);

c. logical (truth tables, boolean equations, symbolic representation, etc.); and

d. control (issue #, vendor #, company part #, approved project usage, etc.).

Relation Scheme Details consist of all the accumulated information concerning a particular Relation. To belabor the point, the Relation Scheme Details' definition consists of a generic description. But, in actual usage, when the key values have been specified for an occurrence of a Relation, then the actual values of the uniquely specified Relation will be generated. Examples of these Details and their two levels (generic and specific) can be seen in the Design Task Data Diagrams (Appendix A).

5. Generic Data. There is an important, inherent characteristic that is defined in the canonical schema and its different relation schemes. The data required by a task

to perform a user's design activities can be generically identified and defined. This means that, regardless of the APs used to perform the design tasks, programs doing smaller functions refer to the same (generic) data, but in a different format. That is, different APs performing the same design tasks need the same (generic) input data.

Different tasks use much of the same generic data, but the tasks will use different names and formats for this design-specific data (specific data requirements). Examples of the specific design data required for a task are presented later.

In fact, it may be surprising to the reader to see the same generic data that is required for different design tasks in a DA System. The generic data occurrences will be described with the canonical schema and its relations schemes. There will be a reduction in the designer's workload as a result of the generic data concept. This is because data will not have to be redundantly provided to the design tasks. Thus, typing errors that would normally occur will be avoided.

This concept does not define data in specific terms, such as where it is used, what is its value, its specific name, data type, or other AP format requirements. Data items have many characteristics, but each individual AP expects to see only a subset of them. Generic data allows the data items to have many characteristics, and each AP

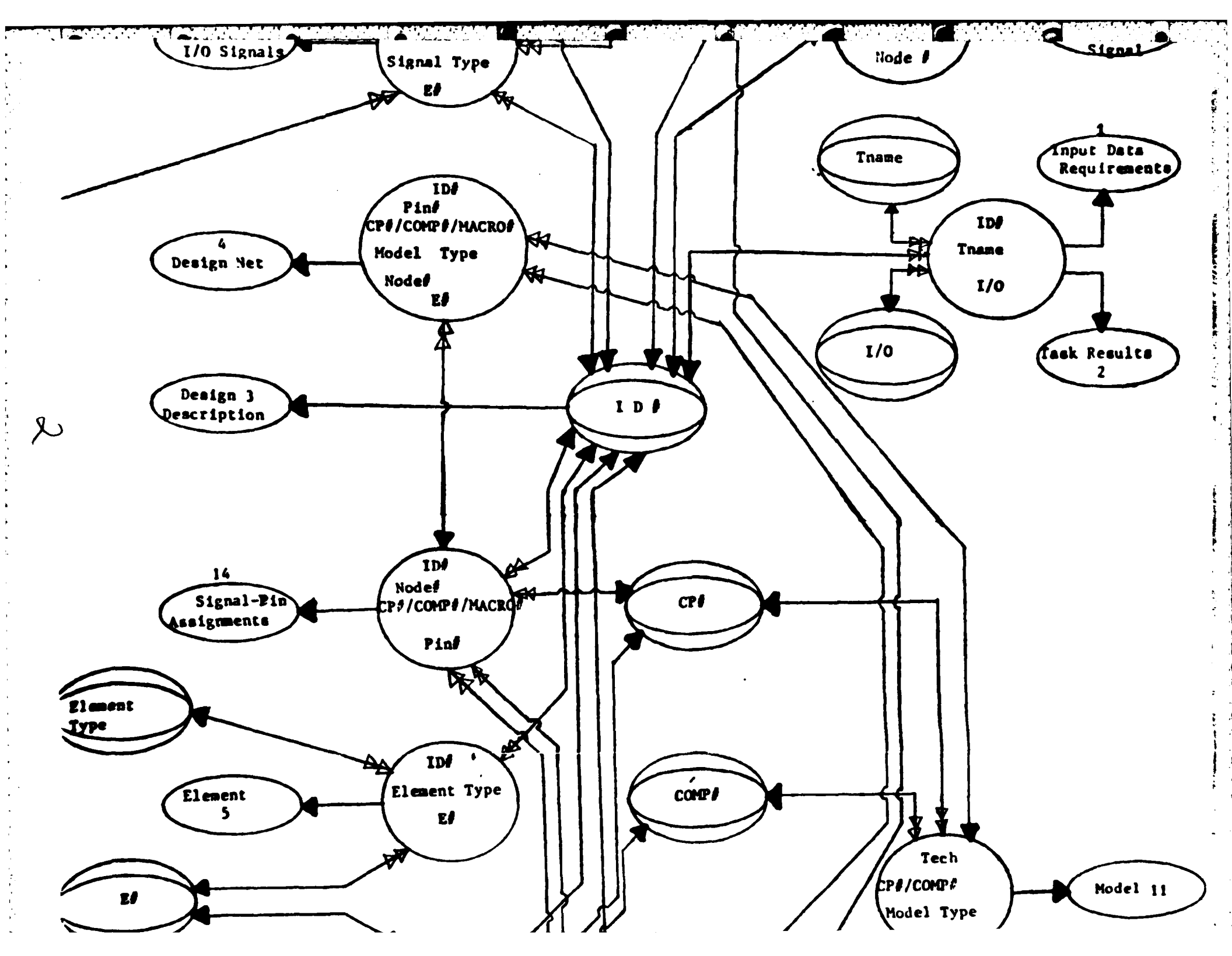
will only see the characteristics that it expects to see. Generic data is defined conceptually and data independent, so it has no physical or logical dependencies. Thus, each AP sees its data as it expects to see it. The generic data must be mapped and transformed for each AP to provide data independence. This mapping and transformation function (performed by the API) is trivial in comparison to the required techniques of a non-integrated data base, and is essential for data independence of the data base.

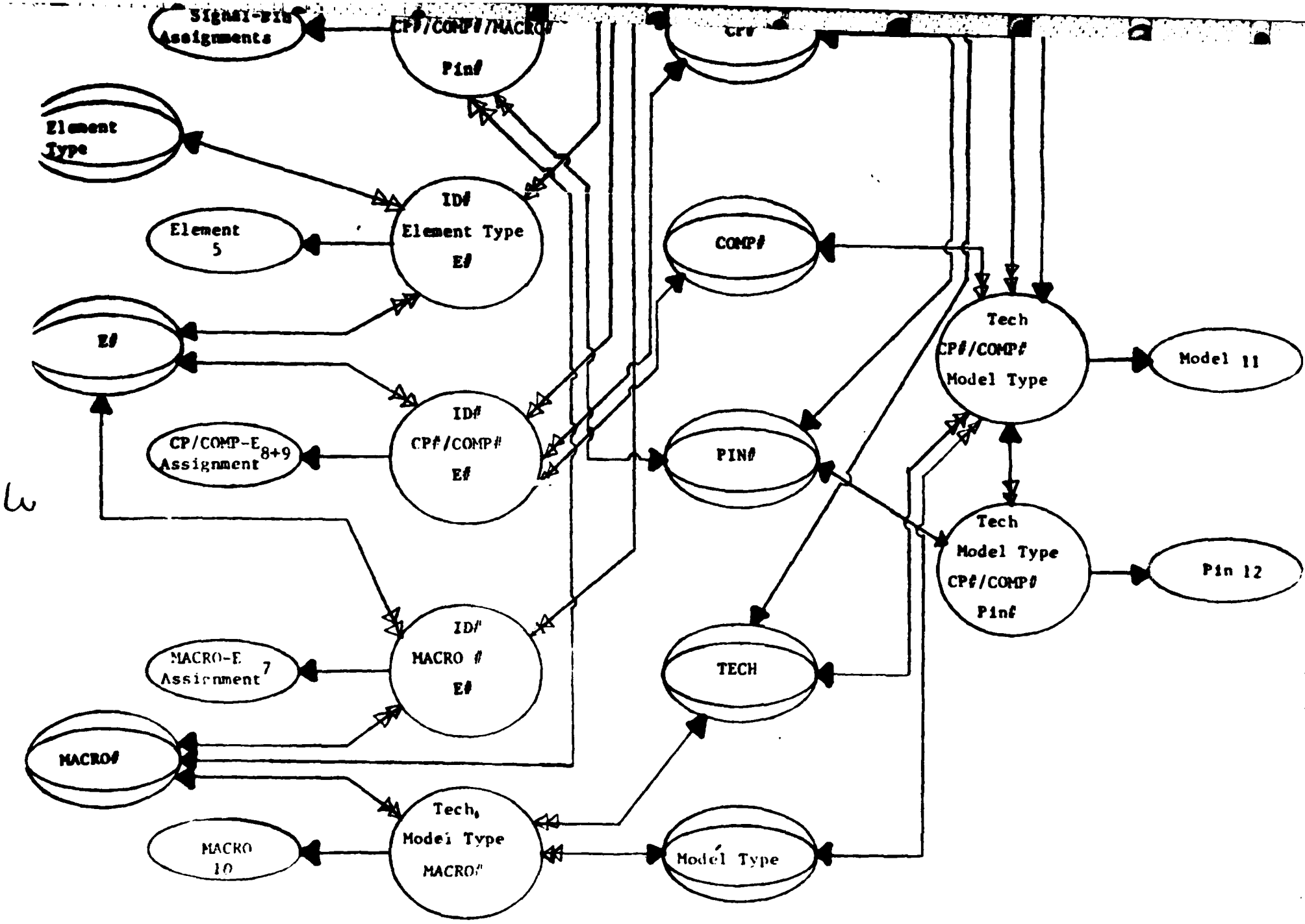
The concept of generic data is a fundamental premise to this thesis. Constructive ways to organize and use this organized data to help design the IDDB is presented later. The use of generic data in designing an IDDB is important because the amount of data normally required for the design cycle is enormous. This report provides an organized view of design data which will facilitate the reduction of the designers' workload on the DA System.

Canonical Schema

Figure 8 shows the canonical schema representing the Integrated Design Data Base. It has been constructed using canonical data structures as described earlier in this report. This data organization model of the IDDB contains a unique characteristic. The data items in this canonical schema are generic in nature. This simplification

FIGURE 8
CANONICAL SCHEMA


I/O Signals
Signal Type
E#
Node #
Signal
Tname
1
Input Data Requirements
ID#
Pin#
CP#/COMP#/MACRO#
Model Type
Node#
E#
4
Design Net
ID#
Tname
I/O
I/O
Task Results
2
Design 3
Description
I D #
14
Signal-Pin Assignments
ID#
Node#
CP#/COMP#/MACRO#
Pin#
CP#
Element Type
ID#
Element Type
E#
Element
5
COMP#
E#
Tech
CP#/COMP#
Model Type
Model 11


Signal-Pin Assignments
CP#/COMP#/MACRO#
Pin#
CP#
Element Type
ID#
Element Type
E#
Element 5
COMP#
E#
Tech
CP#/COMP#
Model Type
Model 11
ID#
CP#/COMP#
E#
CP/COMP-E Assignment 8+9
PIN#
Tech
Model Type
CP#/COMP#
Pin#
Pin 12
ID#
MACRO #
E#
MACRO-E Assignment 7
TECH
MACRO#
Tech
Model Type
MACRO#
MACRO 10
Model Type

characteristic provides the physical and the logical independence required of the design and allows the structure to represent complex relationships.

The design and organization of the canonical schema is discussed here using relational terms and techniques. A description of the canonical schema design will entail the canonical schema, the relation schemes, the information carried within the relation scheme's attributes, and the prime attributes of these relation schemes.

There are two ways that can be used to address the canonical scheme which will minimize the complexity. The first way is to understand the relation schemes, their non-prime attributes (Details and Associations), and the prime attributes of these relation schemes that are contained in the canonical schema. The second way is to look at the data required for the different design tasks, which are seen as task-views of the IDDB. These are described in the Design Task Data Diagrams. Referring to Extended Data Abstraction Hierarchy from the end of Chapter IV, Figure 7 should help the reader visualize how the different views, conceptual and logical, interact to provide a coherent description of the DA System's data requirements.

Understanding the Canonical Schema

Prime Attributes. The following section defines the various prime attributes that will be used to specify the relation scheme keys.

1. ID#. A number that uniquely specifies a design project. This value identifies, in part or in whole, any data that is unique to a particular designer's design. Thus any design input data that specifically applies to the design will have this ID# concatenated with other keys. Referenced data that may be used, but is not design specific, will not need an ID# to uniquely identify it.

2. Element#. Within a design, the Element number is a unique number that specifies each individual occurrence of a CP, COMP, or Macro. Thus, for each NAND gate used, a unique Element# will represent all of the elements' occurrences in the design.

3. Element Type. An element type identifies the specific type of element use or category that the element belongs to. Some of the categories include: circuit, generator, and memory. Circuit elements make up the actual design circuit. Generator elements are used in simulations to generate input signals; they do not become part of the actual design. Memory elements are a special case of elements that often require initialization prior to design simulation.

4. Node#. The node number uniquely specifies each signal within the design. Even though several elements may have the same signal (Node#) as an input each signal is the result (output) of only one element. A node may be represented in two ways: logically (signal) or physically (interconnect-signal).

5. Signal Type. These are categories that different signals belong to, but the signals contained in the Signal Types are not mutually excluded. Some of the categories include Inputs or Outputs, Samples (signal used in timing diagrams), Critical Path designations, Faults (signals to be faulted), and Case Analysis (signals used in timing analysis).

6. Clock. This defines the system time during some analyses. It also provides clock assertions, skew, and general time delays. The time delays are used to calculate interconnect time delays, provide gate delays, and many other time related data items.

7. COMP#. The COMP# (component) uniquely identifies a component type that will be used to fabricate a PC Board. This COMP will have one function, and it may be used several times in the design.

8. CP#. The CP# is a unique number that identifies a Cell Pattern for use on an integrated circuit. A design may use several, one, or no occurrences of a Cell Pattern for IC design. An example of a cell pattern would

be a NAND gate. The NAND gate may be represented in several ways, using different models and technologies.

9. Pin#. Each CP/COMP/Macro occurrence has pins that are connected to signals. These pins are usually labeled and these nonunique pin#s must be associated to a CP#/COMP#/Macro# to acquire meaning and uniqueness.

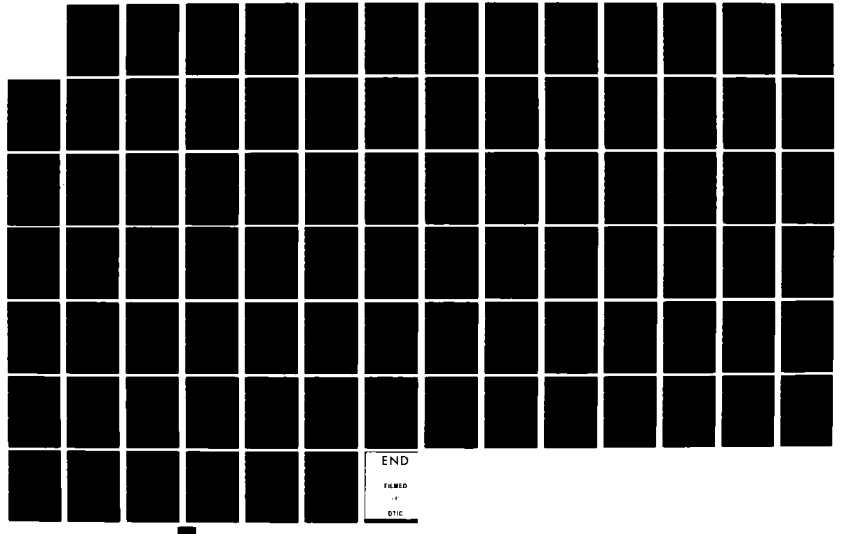
10. Tech. Technology defines the particular technology that will be used during a design implementation. Specification may occur late or early in the design cycle, depending on design requirements. Examples may be CMOS, CMOS/SOS, PMOS/GaAs, TTL, etc.

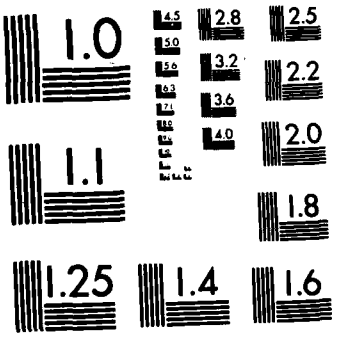
11. Model Type. Every CP/COMP/Macro has several ways that it can be represented and used during the design cycle. These include an outline or interior description, a C/O model, a Timing model, a Cost model, a Circuit model, a Thermal model, and even a model functionally represented as software (Subroutine). This prime attribute must be used in conjunction with a CP#/COMP#/Macro#, and a Tech to uniquely specify a model to use in the design.

12. Macro#. The Macro# is a unique number that identifies a specific upper level function. A Macro is composed of CPs. The Macro is implemented using different Technologies, it has different Model Types, and it also has a Macro Net that defines the specific design, interconnections, and internal and external pins. The Macro Net is described as a Macro Model Type. Because Macros are made

AD-A124 726 MODELS OF AN INTEGRATED DESIGN DATA BASE IN SUPPORT OF A DESIGN AUTOMATION SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M A TEBO 2/2

UNCLASSIFIED DEC 82 AFIT/GCS/EE/82D-35 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

up of CPs, the Macro Model Types are named, for example, C/O Macro versus C/O Model.

13. Interconnect Model Type. A model that represents an interconnect structure that is used to implement the design signals used in the Physical Design Phase.

14. Tname. Task Name (Tname) specifies which design task is to be used. Tnames represent task functions in the design cycle, such as: logic simulation, fault simulation and verification, circuit analysis, IC place, route and artwork, PCB place, route, and artwork, C/O tests, design rule checks, timing verification, pattern generation, and many others. Each Tname has its own set of requirements for input data requirements, and task results as defined by relations of the same names.

15. I/O. This prime attribute defines whether input or output data requirements are needed when specifying the data needed for a design task.

Relation Schemes and Non-Prime Attributes Defined. The following subsections will describe the contents of each relation scheme, its definition, and the non-prime attribute definitions (Details and Association) that are specified by the relation scheme. The relation scheme specification (as seen in Figure 9) consists of the relation scheme name and, within parentheses, the prime and non-prime attributes. The prime attributes are in

"Relation Scheme Name (PRIME ATTRIBUTES..

Non-Prime Attributes..)

Relation Scheme: definition..

Non-Prime Attributes: definition.."

Figure 9. Format of a Relation Specification

capital letters, and the non-prime attributes are not. Also the non-prime attributes are not on the same line as the prime attributes. The descriptive attributes will either be a group of Details or an Association of data items. A definition will then follow of the relation scheme and its attributes.

A relation scheme name is the same as a generic data item used in the Design Data Diagrams. Thus, the reader can easily relate the prime attributes and other information contained in a relation scheme to the information that a design task requires.

1. INPUT DATA REQUIREMENTS (TNAME, ID#, I/O,
   Design Data Requirements (DDR),
   Control Requirements (CR),
   Default Changes (DC)).

Input Data Requirements:

This relation identifies the data required as input to a design task. The data needed is in three categories: DDR, CR, DC. These requirements are unique

for each task. The data requirements are defined generically in terms of relations. However, these requirements are expanded into specific data requirements as shown in Appendix A.

Design Data Requirements (DDR):

These requirements define the design data, from the designer and design tasks, that describe the actual design for a specific design task. Examples of the design data are: Element-CP Assignments, Design Net, ID Details, Element Details, I-S Details, Signal Details, Signal Value Details, Element I/O Signals, Element-Macro Assignments, Signal-pins Assignments, and others that are more task specific. The DDRs are different for each design task.

Control Requirements:

These requirements specify the control statements needed to direct a design task to execute its design function. These statements control the design task's approach, algorithms, functions to execute, default values, and diagnostics and debug.

Default Changes:

The designer has the option to change the default value of data items which are called Default Changes.

2. TASK RESULTS (TNAME, ID#, OUTPUT,
Design Output Data (DOD),
Warning and Error Diagnostics,
Execution Summary).

Task Results:

This relation identifies the data that is produced by a design task. The three main categories of results are listed above as the attributes of the Details. These Task Results Details are defined generically and examples of the expanded forms are shown in the Data Diagrams (Appendix A). The Task Results are unique for each design task.

Design Output Data (DOD):

The design results of a task as directed by the Control Requirements are defined as the Design Output Data. These results are dependent on the task execution. For example, the DOD from a design rule checker task, includes Design Rule Violations, Net Capacitance, Values, Net Check Warnings, Net List, and a Stray Matter List. While the DOD from a Timing Verification task includes a Timing Check for control signals, Set Up and Hold Time Error, Minimum Pulse Width Check Results, I/O Signal Values, and the Design Cycle Time.

Warning and Error Diagnostics:

These diagnostics occur if the control parameters provided are incorrect or there are errors in the design data that prevent execution of a design task. The diagnostics are used by the designer to correct the input data (design or control) to the task to get a valid task execution.

Execution Summary:

Execution summaries are provided after a design task executes. These summaries often include a reiteration of the input data (design and control) provided to the design task.

3. DESIGN DESCRIPTION (ID#,
Design Description Details).

Design Description:

This relation specifies the Details describing a particular design. It describes the specific design characteristics that a designer or a design team has so far designed. Note that all design-related information uses the prime attribute, ID#, as a key, which specifies its designer.

Design Description Details:

These attributes describe the design as it progresses through the design cycle. Any relevant historical

or descriptive data can be stored here. Some examples include revision level of design, technology, designer(s), tasks, algorithms, approaches, processes within tasks used, default changes, design data (verified logical/physical designs), etc.

4. DESIGN NET (ID#, ELEMENT#, CP#/COMP#/MACRO#, NODE#, SIGNAL TYPE, PIN#, TECH, MODEL TYPE).

Design Net:

This relation is often called the net list or the connectivity list. It defines the connections among all the elements (CP, COMP, and Macros) and pins. This net defines an association of all the data items listed as prime attributes. Therefore, there are no non-prime attributes.

5. ELEMENT (ELEMENT#, ELEMENT TYPE, ID#, Element Details).

Element:

This relation identifies the relationship of element# and element type.

Element Details:

The characteristics of each element are described by this relation scheme. The information is dynamically updated during a design task execution or by the designer. These Details can contain placement location, position in

row, row assignment, C/O values, Timing values, or other pertinent data related to an element. Note that an element can be a COMP, CP, or a Macro.

6. ELEMENT I/O SIGNALS (ELEMENT#, SIGNAL TYPE, NODE#, ID#).

Element I/O Signals:

This relation specifies the Signals (Node#), and the Signal Type associated with each element. Usually an element will have several input signals and one output signal. However, the signal types can also be Faulted Signals (stuck at 0/1), or others. There are only prime attributes.

7. ELEMENT-MACRO ASSIGNMENTS (ELEMENT#, MACRO#, ID#).

Element-Macro Assignments:

This relation specifies an association (1:1 mapping) of an Element and a macro. This association is specified by the designer. There are only prime attributes.

8. ELEMENT-CP ASSIGNMENTS (ELEMENT#, CP#, ID#).

Element-CP Assignments:

This relation is a defined association (1:1 mapping) of an Element and a CP, as specified by the designer. There are only prime attributes. Note that Relations 8

and 9 are represented as only one relation in the canonical schema.

9. ELEMENT-COMP ASSIGNMENT (ELEMENT#, COMP#, ID#).

Element-COMP Assignment:

This relation is a defined association (1:1 mapping) of an Element and a COMP, as specified by the designer. There are only prime attributes. Note that Relations 8 and 9 are represented as one Relation in the Canonical Schema.

10. MACRO (MACRO#, MODEL TYPE, TECH,
Macro Details).

Macro:

This relation uniquely defines a Macro as specified by its three keys.

Macro Details:

Macro Details contain information describing the permanent and default information for each Macro. This includes Controllability/Observability (C/O) equations and parameters, Timing Values, Time Delay Values, Subroutine Code and Parameters, Design Rules, and the Macro Net. Macro Detail descriptions can encompass Macros constructed of C/O models, Timing models, Thermal models, Subcircuit models, CPs, COMPs, Cost models, etc. The following examples show how the relation may occur in practice:

C/O MACRO (C/O#, C/O, CMOS/SOS, details),

TIMING MACRO (TIMING#, TIMING VERIF., TTL, details),

SUBCIRCUIT MACRO (SUBCIRCUIT#, SUBCIRCUIT, PMOS, details).

The details are not specified in the relation since they are the data that is generated by the specification.

11. MODEL (MODEL TYPE, CP#/COMP#, TECH, Model Details).

Model:

This relation defines specific information on each representation of a CP/COMP. This is directly parallel to the Macro Relation as described above.

Model Details:

Each CP#/COMP# has several model representations. Each pair of Model Type and Tech defines a different group of CP/COMP Details. Some of these model's Details include: outline and interior geometries, Design Rules, C/O models, logic symbol, cell heights, engineering revision level, connection logic, timing models, circuit models (source, gate, drain, etc.), Cost model, Thermal models, etc.

The following are examples of model occurrences or relations:

C/O MODEL (C/O, 7400, TTL, C/O details),

CIRCUIT MODEL (CIRCUIT, 25, GaAs, model equations,

electrical characteristics, initial conditions), CP OUTLINE DESCRIPTION (CP OUTLINE, 7954, ECL, outline description, outline design rules).

12. PIN (PIN#, CP#/COMP#, TECH, MODEL TYPE, Pin Details).

Pin:

This relation defines pins that are on CP and COMPs. Each CP/COMP has several pins and each pin has certain characteristics that are defined in Pin Details. Note that it takes four prime attributes to specify these Details.

Pin Details:

Pin details include pin type (I/O), pin impedance, node capacitance, and physical characteristics, such as thermal flexing and pin size.

13. SIGNAL (SIGNAL TYPE, NODE#, ID#, Signal details).

Signal:

This relation defines a signal used in a design net.

Signal Details:

These describe a signal's characteristics, before it is physically implemented, used in the logical design nets.

14. SIGNAL-PIN ASSIGNMENTS (NODE#, PIN#, CP#/ COMP#/MACRO#, ID#).

Signal-Pin Assignments:

This relation defines the association of CP, COMP, and Macro's pins with their associated signals. This association is specified by the designer. There are only prime attributes.

15. SIGNAL VALUES (MODE#, SIGNAL TYPE, CLOCK, ID#, Signal Value Details).

Signal Values:

This relation defines the value of a signal, at a particular time. These values have many ways that they can be calculated.

Signal Value Details:

The signal values and techniques to calculate them are contained herein. Thus, these details contain the values of each signal at specific clock time. Sometimes these signal values are calculated according to combinational logic tables (of which there can be several), to a formula, of listed, as in the Fault Simulation case. Stable assertions (when control or data signals are stable and when they will change) are defined. Other values are also defined, such as Signal Skew, Values Width, Rise and Fall Times, etc.

16. INTERCONNECT (INTERCONNECT MODEL TYPE, TECH,
Interconnect Details).

Interconnect:

This relation relates an Interconnect Model type with a Technology which specifies its details. Note that there is a general interconnect description which is the model that is to be used when implementing a signal. It is not, however, the actual implemented signal.

Interconnect Details·

These Details include specific electrical characteristics and design rules that exist for a specific technology of interconnection. Other details include spacing, maximum line length, line width, number of bends allowed, technology, signal crosstalk, and reflection values, and other design rules to be used in the implementation.

17. INTERCONNECT-SIGNAL (INTERCONNECT MODEL TYPE,
NODE#, TECH, ID#,
Interconnect-Signal Details).

Interconnect-Signal:

This relation requires four prime attributes to specify it. It defines the actual implemented interconnect structures used to make a signal.

Interconnect-Signal Details:

These details describe the specifics of the implemented signal, such as signal coordinate route, electrical characteristics, line length, time delay, etc.

18. CLOCK VALUES (CLOCK, ID#,
Clock Value Details).

Clock Values:

This relation defines the clock details which describe the clock and time functions over time.

Clock Value Details:

These details contain clock-related information, such as clock cycle time, basic time unit, minimum pulse width, constraints, min/max propagation delay of a design, set up and hold time constraints, gate delay, clock skew, etc.

Meaning of the Canonical Schema

The many component parts of the canonical schema have been explained: the Relation Schemes, their Attributes, and the different levels of the data as represented in the Design Task Data Diagrams and the Canonical Schema. These components provide important information concerning the data base, the data it will contain, and the relationships among the data.

The Canonical Schema's purpose is to represent the total view or the "map" of the data base. It is to be used

as a reference tool to navigate through the data base. As seen in the Data Diagrams, specific data is not found in the canonical schema, only a high level representation of the data.

This high-level view of the data base is absolutely necessary for the next two steps for the design and implementation of the IDDB. The Canonical Schema fits between the results of these two schemes and provides a common map to which they can both refer. The physical and the logical views of the data are diverse; this map facilitates and guides the required design and implementation procedures.

In summary, the Canonical Schema is essential to the successful design of a data base model, especially the Integrated Design Data Base. It not only provides an excellent map of the overall design and of the data, but is the key solution to the critical goals of physical and logical independence. Also, because of the normalization process the Canonical Schema was designed with, the Canonical Schema will avoid anomalies and will readily accept the changes and growth of the data base over time.

CHAPTER VI

CONCLUSIONS

This report has addressed the problem that is faced by microelectronic designers when they find the computer tools available to them for design work are difficult or at least not easily used. The objective of the work that has been described in this thesis report has been to improve the usability of certain computer tools which, in particular, are the DA System and its Integrated Design Data Base. The overall goal of this thesis has been to develop a conceptual-level model of a DA system including an Integrated Design Data Base.

Six Goals Discussed

The overall goal was broken into six individual goals. These six goals and how they were achieved in this thesis report will constitute the remainder of this chapter.

1. Discuss the background of the problem and the essential elements of the solution. This was done in Chapters I (Introduction) and II (Background). In Chapter I some important concepts such as data independence, logical and physical independence, and logical, physical, and conceptual schemas were discussed. Also, design considerations and the purpose of the report were discussed.

Finally, the six goals that are now being summarized were introduced.

In Chapter II a background is provided for the problem areas that this thesis report addressed and the elements of the solution that would provide the potential techniques and concepts essential for completion of the thesis objective. The essential components, techniques, and concepts of the solution are Software Engineering and Data Base Design Techniques, DA Systems, and an Integrated Design Data Base. Thus, the first goal of the thesis has been satisfied.

2. The software engineering and data base design techniques and design approach must be described. This second goal's results were described in Chapter III (Design Overview). The main techniques used were Design Task Data Diagrams, Design Cycle Activity Diagrams, Third Normal Form, Canonical Data Structures, and Structured Walk-throughs. Each of these techniques was discussed and the process required to perform each technique was described where necessary. In achieving this second goal, the Design Approach used to perform the design work was also described. The Design Approach consisted of four steps that when completed would satisfy all of the six goals of this thesis. The four steps were to:

a. collect the user's design requirements;

b. collect the total system requirements;

c. build the canonical schema; and

d. document the data organization, in terms of requirements, for both the users' and the system's views.

As can be seen from Chapter III, the second goal has been satisfied.

3. Present and characterize the important, known design task data requirements of the DA System. The results of this goal are the specifications of the requirements of the Design Cycle, design tasks, and the individual data requirements of these design tasks. These requirement specifications have been essential to the development portion of this report. Some of the results can be seen in the Design Task Data Diagrams and the Design Cycle Activity Diagrams. Therefore, the results of this work are the foundations that the DA System and the Integrated Design Data Base models have been built on. This is the area of research that is logically and physically dependent on the specific hardware and software that the design system is to be constructed. Thus, when further design and implementation activities are initiated, this will be the place for these requirements specification activities to begin.

4. Design the conceptual DA System model which will be used as the model to describe the IDDB environment. This was accomplished in Chapter IV (Design of the DA System model). This goal not only entails a high-level

design of the DA System, but also a description of how the IDDB will be used within this system design. The description of the DA System contains a group of general assumptions about the system, the organization of the system, and how the system will be used. The organization of the system lists and describes the components that will be used in constructing the system design. A discussion of the designer's point of view during the use of the system is provided along with a more detailed description of the internal interactions of the DA System components as they perform their required functions. A final discussion is included concerning the Design Cycle. The main purpose of this discussion is to show the interactions of the designer, the design data, and the DA System during the long, complex process of the Design Cycle. The discussion also shows how certain characteristics of the DA System can be used to satisfy the original objective that is being addressed by these six goals: to increase the usability of the computer tools available for use by the designer. The fourth goal has been accmplished and has laid the groundwork for the fifth goal.

5. Design the conceptual-level model of the integrated design data base (the canonical schema) which is physically and logically independent of hardware and software considerations. This fifth goal's achievements were described in Chapter V (Design of the Canonical Schema).

This chapter describes the model, the Canonical Schema, which represents all of the design data required by the IDDB for use in the Design Cycle. The components of the design are presented and discussed. These components are represented in diagrams and in a relational data model format. These relational representations include the relation schemes, prime attributes, non-prime attributes, and descriptions of all these representations are provided. After the elements of the Canonical Schema were discussed, the meaning of the Canonical Schema was presented. These discussions thoroughly presented the model of the IDDB and described how the conceptual model can be used to map the specific data requirements of these design tasks onto the logical requirements of the IDDB. Because of the techniques used during the design process of the Canonical Schema, the results are physically and logically independent of any hardware and software considerations. The fifth goal has been accomplished as described in Chapter V.

6. Provide recommendations concerning the implementation and maintenance of the DA System and its IDDB. The results of this goal were presented in the next chapter (Recommendations). These recommendations included a data model choice and the justification of the choice. It also provided a discussion of the required characteristics of the IDDB and the DBMS that will manage it. The next section discussed an implementation plan that provided a

strategy of how to most effectively complete the design and implementation of the models of the DA System and IDDB. These "challenges" constitute the implementation plan and are broken into segments that are the size (effort-wise) of a class project or of a thesis. Finally, there is a detailed discussion of the functions, activities, and tools required of the Data Base Administrator. Thus, Chapter VII provides a complete discussion that satisfies all of the requirements of the sixth and final goal. Therefore, with the completion of this chapter, the objective and the overall goal of the thesis have been met and have been documented in this report for future use and constructive discussions.

Final Concluding Remarks

There are three reported accomplishments achieved in this thesis report. The first two accomplishments are the models that were developed: the DA System and the Inegrated Design Data Base. While the author knows of no flaws in the models presented, he does not totally reject that possibility, since it is an untested model. At the worst, the models can be used to stimulate interesting discussions and debates. However, the author does feel that these models do efficiently and effectively describe a useful contribution for designers of microelectronics.

The accomplishments of the two models are useful to the DA field in general, but the third accomplishment is relevant specifically to organizations, such as the Air Force Institute of Technology, who will implement the models. Thus, the third accomplishment is the implementation plan provided in Chapter VII. If the steps are followed and the functions of the components are built and tested as designed and described, then the result should be an efficient, flexible, and user-friendly DA System.

CHAPTER VII

RECOMMENDATIONS

Data Model Choice

The three major data models are the Relational, Network, and Hierarchical data models. There are many books written that describe these data models in great detail; some of these are References 25 to 29. Several factors were used when considering the data models and their effects on the chosen data model for the implementation. The primary factors that were used were usage effectiveness and implementation efficiency. As a result of using these factors to choose a data model, the relational data model was found to be superior and was chosen.

Usage Effectiveness

This factor measures the ease in expressing a query to operate against the data base. This factor also measures the (data) manipulative ease. The term "manipulative ease" implies that there are a small number of operators and that there are high level operators available. Thus, not only must the queries be easy to express, but they must be accurate.

Implementation Efficiency

This factor is used to consider implementation difficulties and advantages concerning hardware and software efficiency that is affected by a characteristic of the data model. Storage space for the data structures and computer time for processing queries are major factors that dominate the implementation cost of a data base.

Justification of Choice

Disadvantages. The areas where the relational model is weakest is in implementation efficiency. The relational data model does not presently perform well on large data bases. However, the physical implementation of relations (tuples) is much less complex than tree and network structures and access strategies. Therefore, it is slower but less complex to design, implement, and maintain. Also, the apparent inefficiencies mentioned above are being eliminated through research. For example, JOINs should not usually be physically performed, but instead should simply provide a logical view of the results of the JOIN. There are many techniques being developed and implemented that improve the present inefficiencies of relational DBMS. One category of techniques that has a lot of potential is query optimization.

Many of the efficiency problems that do not occur in the other data models, but do occur in the relational

data model, have been previously solved in the other models. Research in the relational model area is proving that there is reason for optimism for a relational data model implementation that is comparable to the other models in efficiency.

Advantages. The relational data model has many good points to consider when evaluating its advantages. As far as user effectiveness, the relational data model uses only one construct, the relation. The query languages, used in relational data manipulations, are rich, high-level, and easy to use. Thus, the relational data model scores very high in user effectiveness.

When representing the relational model in Third Normal Form, there are many other advantages. Some of these are listed below:

1. Ease of Use. Relational queries are very easy to use for all levels of data base expertise.

2. Flexibility. Relational operators support flexible data base operations.

3. Precision. "The precise results of relational mathematics can be applied to the manipulation of relations [25:226]."

4. Security. "Security controls can be easily implemented [25:226]."

5. Ease of Implementation. Tuples, or tables, are easily implemented.

6. Data Independence. Data independence has been previously discussed.

7. Data Manipulation Language (DML). Provides the flexible specification language used for queries. Can be based on relational algebra or relational calculus.

8. Clarity. The relational data model is easily understood, mainly because of the simplicity of the data structure and the mathematical logic of its DML (25:226).

An example of a query optimization technique is a program that will perform a logical optimization on a query, breaking the query into several smaller queries. The result is efficient because huge parts of the tables that would normally have to be constructed have been deleted by proper query specifications. Frequently used queries are usually the best choices for optimization. Another example of an optimization technique is to physically store frequently used relations so they are readily accessible. This is often called data migration. As has been described by these optimization techniques, there is a lot of potential for improved efficiency as a result of having queries and data accessibility optimized.

Another advantage of the relational model is the attention it is receiving from researchers. The cause of the attention is for three main reasons. First,

relational inefficiencies are less obvious in small and medium sized data bases and most existing data bases range in size from small to medium. Second, it has been discovered that most of the inefficiencies of the relational data model can be eliminated. The third reason is the most important. The relational data model has a sound mathematical basis that is rich in theoretical and applied research topics. Improved optimization techniques and other research into the potential of this mathematically rigorous data model and its data manipulations all hold a great deal of hope, interest, and progress for this area.

The functions and characteristics of the DA System, using the IDDB, provides an excellent environment for a relational data model implementation via a relational DBMS. The inefficient part of the relational data model, i.e., queries against a large data base, are minimized because small to medium sized data bases are mainly used; that is, Project Dependent and Execution data areas are not large. These smaller data bases will contain the data that is manipulated most by the DBMS and used by the Application Programs. The most frequently used data access paths (for this report's design) have been identified:

1. reviewing the design data,
2. generating the PD and Execution data areas, and
3. storing new design data into the PD data area.

The DA System's functions and characteristics are ideally suited for the relational data model and especially for optimizing techniques that can be implemented within the system.

Also, the flexibility of relational queries will be available for use with the small data bases in the PD and the Execution data area. Generating the data in the Execution data area requires data queries against a large data base. These queries against the large (Project Independent) data base are prime candidates for optimization techniques. Thus, with careful planning and implementation of optimizing techniques, the DA System users can enjoy the usability and flexibility advantages of the relational data model without being negatively affected by its present inefficiencies.

A final selling point in the favor of the relational data model is that it naturally supports a High Level Language (HLL) which is crucial for future engineering tool developments. The relational model can support such a HLL because it has a sound theoretical basis, presents a simple interface (one data type), and has a natural language-like navigational language (versus record-at-a-time). The relational data model is important for future development of engineering applications because it will allow interaction between the data base, the artificial intelligence, and the engineering community (36:866).

Briefly, in summary, the relational data model is very effective to use, but the other two data models at the present time are more efficiently implemented. The many advantages of the relational data model and relational DBMS outweigh its minor inefficiencies. Finally, the characteristics of the IDDB environment will favorably use the relational data model to great advantage.

Characteristics of an Integrated Data Base and Its DBMS

Objectives

The ultimate objective of an integrated data base is to make application program development (i.e., I/O data manipulation) and systems integration (with the data base the kernel) easier, cheaper, faster, and more flexible. The system must be usable; therefore, it should simplify the designer's work. Reliability is an important consideration also. The system should be available when it is needed and should not fail while it is in use.

An integrated data base contains the data needed for the system's data processing. That data should be accurate, secure, maintained, and protected from misuse and unauthorized change. The data organization should help users with different applications which have different data requirements. The overall data organization should allow different APs and designers to have different views of the same data. This is especially important

for these two cases of changes: (1) when existing APs may be changed to process the data differently, and (2) when new APs will be integrated into the DA System which require new views of the data. The costs to make these changes should be minimized, and the changes should not affect other logical views of the data or the physical view of the data.

To achieve the above objectives the data base and the DBMS must have certain characteristics. When these characteristics are designed into the implementation of a data base organization and DBMS, then its ultimate design objective can be met. It is also important that the data base and DBMS characteristics define a dynamic and flexible system that will absorb change and be able to be used efficiently and effectively.

Characteristics

The organization of an integrated data base and the functions of its DBMS should provide certain desirable characteristics. These characteristics are briefly described.

1. Ability to represent the inherent structure of the data and define abstract data types. The relational data model and its DBMS implementation contain these characteristics.

2. Simplicity in describing the overall logical structure of the data base. (A canonical schema is an ideal description.)

3. Integrity for data items and associations between them must be maintained. Data integrity should be provided during storage of data, data updates, data insertion, and system failures. Thus, integrity implies that all calculations should be carried out properly and produce correct results. If the relational DBMS is correctly designed, implemented, and tested, then all of these characteristics will be satisfied.

4. Interface with the future. In the future the data, its storage media, and its usage will change. It is critically important to design the data base such that these changes will not require changes to other data or to the APs that use the data. Data independence is the key in providing a physical and logical buffer against change. Thus, physical and logical data independence will help preclude the possibility of such drastic effects on the data or APs. This characteristic (data independence) is the most subtle error possible, during the design and implementation phases of the data base, but the most obvious afterwards. The models described in this report are data independent.

5. It is important to minimize redundant data storage to help prevent anomalies that can occur with

redundant data used in data manipulations. The relation schemes contained in the Canonical Schema which are in Third Normal Form and by definition are minimized and normalized.

6. The system's performance should exhibit acceptable efficiency in carrying out the desired computations and provide reasonable response times which are appropriate for the person-machine dialogue. Response time will usually depend on the traffic volume, the data base physical organization, the query, and the hardware capabilities.

7. It is also important to minimize the cost of operations through data migration and tuning the system to the individual requirements of the system users. Trade-offs exist to minimize the storage requirements, while maximizing data accessibility. The DBMS must take data migration into account (difference in frequency of use of data items) and, in response to data migration, tunability of the system (adjust physical view or optimize queries), when attempting to minimize the cost of data base operations.

8. Data security and privacy are important considerations.

> Data security refers to protection of data against accidental or intentional disclosure to unauthorized persons, or unauthorized modification of destruction. Data privacy refers to the rights of individuals and organizations to determine for themselves when, how, and to what extent information about them is to be transmitted to others [25:38].

9. A powerful user language including a search capability, is needed. A powerful and flexible user language will allow easy access to and manipulation of the data base. This will be a great help in the development of the EXEC, AP interfaces, and the DBMS Interface. The query language is usually built into the DML of a relational DBMS.

(Characteristics 1-9 are from Reference 25:34-47.)

10. The DBMS should provide an interface to an HOL (Fortran, Pascal, AdA, etc.).

11. The DBMS should support multiple design representations.

For the DA System being designed, the most important characteristics of the data base's design are physical and logical independence. These characteristics will allow the data base and the APs that use the data to evolve and change without affecting the overall data base organization or the APs in the DA System.

Implementation Plan

There are five "challenges" that exist to be addressed and resolved to be able to implement the DA System and its Integrated Design Data Base. These challenges are described in the following paragraphs.

Challenge 1

First, several APs must be chosen that will be initially integrated into the DA System. It would be helpful and logical if these APs made up one of the six phases of the design cycle previously discussed in Chapter IV.

Second, the task-views, or logical schemas, of the chosen APs must be designed, which will include the input, output, control data and data formats. These results will be defined as relations and will be subsets of the Input Data Requirement and the Task Result relation schemes (which have been previously described in Chapter V). Next, an AP Interface for each of the APs to be implemented must be written. The API will contain the relation definitions. The API must also contain handlers which "parse" the control commands issued from the EXEC and the Data Dictionary.

Challenge 2

A Relational DBMS must be chosen and implemented that satisfies the characteristic functions of a good DBMS, previously listed. There should be effective tools for the DBA to use (DDL and DML) to create, load, and change the integrated design data base. The Project Independent (PI) data base must also be loaded with the minimum required data required to run the chosen APs. After the APs are chosen and the APIs are complete, then the Data

Dictionary must be written. The DD contains the logical location, definition, and organization of all the data in the IDDB.

Challenge 3

The DBMS Interface must be developed which will issue commands to the DBMS and pass responses back to the Data Dictionary. It must be able to:

1. Receive relation specifications from the Data Dictionary;

2. Convert these relations into acceptable DBMS commands;

3. Issue commands that create, find, read, write, test, and change the Project Dependent data area; and

4. Issue commands against the Execution data area that find, create, read, write, and delete from Execution data area.

In particular, the test operation queries the PI and the PD data areas if required design data is available; the read operation allows the new design data to be shown to the designer; and the write operation allows the design data to be written into the PD and the Execution data areas.

Challenge 4

The Executive must be designed and implemented. First, the design requirements of the EXEC should be specified by the DBA prior to the EXEC design effort. These

design requirements will describe the DA System capabilities, present and near-future, that should be supported by the EXEC.

The initial design should be modular and hardware independent. The design implementation of the EXEC should also be very modular. The exceptions of the hardware independence requirement must be carefully modularized and must be documented. The EXEC code written that is not hardware dependent shall be written in a common standardized HOL, so it can execute on other computer systems that support a compiler in that language. Thus, the EXEC will be able to run on different hardware with only the hardware dependent modules having to be rewritten.

Some of the functions to be implemented into the EXEC should be:

1. The EXEC should be able to identify which APs are to be run and who (i.e., designer ID#) will run them.

2. The EXEC should also be able to query the data areas (i.e., PI, PD, and Execution) through use of the Data Dictionary, and allow the designer to verify the design data and the default parameters of the design tasks.

3. The EXEC should permit the designer to be able to save data generated by task executions and route it from the Execution into the PD data area.

4. The EXEC should also permit the designer to be able to delete data from the Execution area, change

data in the PD area, and make Default Changes through the use of the EXEC.

5. The EXEC must provide "automatic management of precedence ordering" to control data dependencies (1:1254). As previously discussed in Chapter IV, there must be some way to implement the required control of the data dependencies, as implied in the Design Cycle Data Diagrams. One approach is supported by Charles Eastman who discusses the use of transaction graphs which seem to have potential to help monitor data dependencies (1:1254-1255). The information contained in low-level Activity Diagrams can be directly translated into the required transaction graphs. Then the graph must be compiled into a form usable by the EXEC. Once these steps are done, the resultant process will monitor and control the data dependencies.

Challenge 5

The goal of this challenge is to provide a continuous update of the previous four challenge activities and coordinate these changes so that the system is maintained. Thus, this is a continuous challenge to design and implement the capability to manage the growth and modifications of the DA System.

Summary of Challenges

Challenges 1, 2, and 3 can be performed simultaneously if designed carefully and the results of each are coordinated. The second challenge needs to receive the subsets of the relations that define the Input Data Requirements and Task Results from the first challenge. So, a modular, standardized interface must be defined between the API and the Data Dictionary. Also, a modular, standardized interface must exist between all the DA System components in Figure 5 at points:

Designers/EXEC	1/18
EXEC/AP Interface	2/17
AP Interface/Data Dictionary	3/14
Data Dictionary/DBMS Interface	4/13
DBMS Interface/DBMS	5/12
Data Dictionary/EXEC	19/20

The overall objective of the design and implementation of the DA System and its IDDB is to provide a flexible, usable design tool for the designer to use, that will allow growth and change of the components of the DA System.

Data Base Administrator (DBA)

The term DBA is being used loosely in the following discussion. Instead of just managing the data and the DBMS, the DBA functional description is also managing the entire DA System. So the term used would more properly be a Systems Administrator, but DBA will still be used.

DBA Qualifications

The DBA must be able to understand the problems of the users of the system, the designers. The DBA must also be able to evaluate, through software engineering techniques, the efficiency and effectiveness of software design and code that will become the DA System. The DBA must be technically competent to understand the present use and implementation of the DA System, and to plan for future growth and changes to the System. The DBA must have the authority to handle designer's and system problems. This implies that the DBA must be able to obtain or allocate data collection, programming, computing, and communication resources (26:596-698).

It may be obvious to the reader that a DBA is usually not one person, but is a team and the DBA is often referred to as a DBA Function, thus allowing these qualifications to be spread among several people. The following paragraphs will discuss the various functions and responsibilities that make up the DBA Function.

The following two sections will describe the typical DBA functions and then the DBA functions that are required during the system design and implementation will be separately discussed.

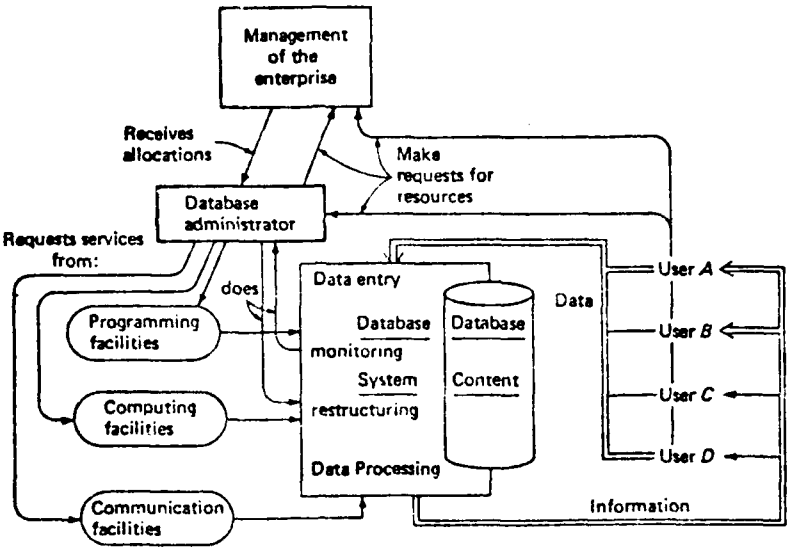
DBA Functions

The DBA maintains the overall structure of the data. The DBA is the custodian of the data, but does not

own the data. For this report, the DBA controls the PI data and maintains the organization of the PD data. The DBA must guide and plan the course that the data base will take in the future. Therefore, the DBA must have a complete understanding of the data base, its organization, its economics, its design criteria, its change and growth patterns, and the needs and requirements of the designers and the design tasks that use the data.

Figure 10 shows the DBA location within the DA System's environment. The users are the designers using the DA System. The Programming, Computational, and Communication facilities are some of the resources that the DBA function manages. The DA System is not shown, but the IDDB component is represented as the "Database Content." The "Management of the Enterprise" for this report represents the needs of the AFIT educational and research community (26:596).

Responsibilities. The DBA's responsibilities include generating the information content, the storage structure, and access strategy of the data base. The required information content has been defined at the conceptual-level by this report. The DBA must decide, by choosing APs which will be integrated into the DA System, the final design data to be integrated into the IDDB. The storage structure and access strategy have already



A place for the database administrator.

Figure 10. "A Place for the DBA" (26:596)

been partially specified because the relational data model has been recommended, the conceptual schema has been defined, and an implementation plan provided for the logical and physical schemas. The DBA function is responsible for designing and implementing data independence and data integrity into the data base. Data independence is achieved when the requirements of a design task (logical schemas) are revealed to the designer and the storage representation (physical schema) used in the implementation is hidden. This also implies that any changes to the physical schema or to the logical schema will have no effect on the other schema.

The DBA must define a strategy for backup and recovery. A backup schedule must be identified and enforced by the DBA. The recovery strategy must describe

the actions to minimize damages caused by human error, or hardware and software failures. The system and data should also be repaired with as little effect as possible on the rest of the system.

Set Goals. The DBA should also set goals (i.e., policies, standards, and procedures (32) for the system. For example, these may include:

1. Response times--i.e., 90 percent of single-element queries should take less than five seconds between query entry completion and beginning of response.

2. Backup--i.e., backup will be available for all design data results that have been saved for an hour and all input design data entered at least four hours ago should also be backed up. Also backup for all deleted data up to two weeks ago.

3. Deadlock--i.e., there should be less than two deadlocks per year.

4. Size--i.e., there should be the capability to access a certain number of data items and be able to process designs of certain sizes (number of active elements approximately 100K, etc.).

There must also be documentation and configuration control of the design data (PD and PI data). The DBA must make sure that the data base representations (conceptual, logical, and physical) are well documented. This

documentation is absolutely vital for the DBA's responsibilities. Documentation of the standardized interfaces between DA System components and the hardware dependent code module documentation is essential to the life of the data base and the system in which it resides.

The final point concerns the responsibilities that accompany the role of the DBA. The DBA has access to all of the data in the system and to all the software of the system. These privileges provide the DBA with a view of the system that the designer does not see. It is a total system view, and it can be seen at all three levels: conceptual, logical, and physical. While it is necessary that the DBA have these views and capabilities, it is also necessary to have some system of checks and balances among the members of the DBA team. Trust of the DBA is necessary, but it is also important that no one person has absolute power. There are many ways to deter abuse, such as preventing the DBA from writing application programs against user data, or by partitioning the passwords and access rights among the DBA team. These considerations should not be extreme in the educational environment of this report. The important factor here is to prevent accidental data corruption.

DBA Tools. The DBA has several specialized tools and utility programs that provide invaluable help to the

DBA Function. Some of these routines include:

1. Loading routines, which create an initial version of the data base.

2. Data base compaction routines, which are useful for reducing unused data or obsolete data in the (PD) design data areas.

3. Journaling routines, for both the data base operations and EXEC (designer) operations. These are good for historical reference, data base reconstruction, and statistical analyses.

4. Recovery routines that restore the data base after hardware and software failure.

5. Statistical analysis routines provide the performance evaluations of the software, hardware, DBMS queries, and optimization techniques (27:27).

6. Report Generation provides routine and extraordinary information required for use by the DBA function (26:465-476).

The Data Dictionary is one of the most useful tools the DBA can have. It contains the descriptions of all the data in the data base. Also, the conceptual, logical, and the physical schemas are stored here. (The logical schema is actually implemented in pieces (partitioned) in the APIs, but the overall logical schema description is in the Data Dictionary.) Each task view, or partition of the logical schema, contains the data used

and generated. It also contains the format of the data required for a design task. The Data Dictionary will also contain a cross-reference listing showing which APs use which relations and which specific data items.

The other tools that are used in conjunction with the DBMS that the DBA uses are Data Description Language (DDL) and Data Manipulation Language (DML) supplied with commercial DBMS. These tools are used to create, load, update, and change the data base, especially the PI data area. The DDL provides the means to create the data organization for use by the DBMS once it has been designed. Thus, the DDL will be used to specify the Canonical Schema that has been described in this report. The DML provides the mechanisms for retrieving records from the structure that has been defined by the DDL.

DBA Functions During System Implementation

The DBA function must (repeat must) use software engineering tools and techniques, during the design, implementation, and management of the entire DA System. Relevant software engineering tools and techniques to be used (as seen in Appendix D) can be classified according to the system life cycle: Requirements Specification, Design, Implementation, Testing, Maintenance, and Documentation.

The design requirements for the EXEC should be specified by the DBA. The APs to be integrated into the

DA System and the designer functions to be supported by the EXEC also need to be specified by the DBA. The DBA must verify and coordinate the design and implementation of Challenges 1, 2, 3, 4, and 5. This includes planning and coordinating the DA System changes and growth, as described in Challenge 5. Two excellent references for the DBA managing the development and implementation of these challenges are Productivity in a Data Base Environment (13) and Design Review Methodology for a Data Base Environment (31).

The DBA must maintain and update hardware and software capabilities which includes: (1) acquiring new hardware and software; (2) maintaining the contractual activities of this procurement; and (3) monitoring the student support of software research, development, conversion, and update. (If there exists common hardware and software especially operating system, among other DOD agencies, universities, and research facilities involved with microelectronic design, it would be especially helpful to form an alliance. Then software could be easily shared, and updated software need only be changed once and then distributed. There are many advantages to an alliance such as this, not the least being interchange of knowledge and shared resources.)

The DBA should analyze the performance of the DA System and evaluate the physical schema to see if

frequently used data can be optimized (data migration). For example, response time is a good first-cut evaluation parameter. The DBMS queries and commands can also be evaluated to find candidates for query optimization.

The DBA must, for each designer on the DA System, issue an ID# and verify that a Project Dependent data area is created and available to the designer. Access to the DA System must also be controlled through passwords and/or ID#s.

Once the DA System is up and running, the designers will notice a gap between expected and actual DA System capabilities. In a prioritized manner, these insufficient capabilities should be added or improved to minimize interference; all such additions and improvements should be done in a background mode, which will be transparent to the designers using the system.

Analyses. Depending on the complexity of the DA System, several analyses may have to be performed before and during system implementation. These analyses are briefly described:

1. Operational/Functional Shakedown, which shows the actual capabilities that work is designed. These include validating the functions of each component. It is especially important to validate the DBMS read, write, and change functions, because if the system will not

support reliable data operations it is worthless until corrected.

2. Verification of Data Content after the data has been integrated. The designers using the DA System must be able to be confident that the data stored in the IDDB will retain its integrity. The data must also be reverified after several tasks execute, and thus use, store, and generate design data. This will help ensure that the DA System component actions are correctly integrated and thus will maintain data integrity.

3. Monitor the System Performance by performing systems analyses and performance evaluations. Statistical analyses of data contents, system activity, response times for queries, and data storage requirements should also be generated. Query optimization techniques also need information concerning utilizations of devices, relation schemes and relations. These evaluations will keep the DBA alert for new designer needs, such as additional or changed design tasks or EXEC function, and other evolutionary effects on the DA System's environment. This monitoring function aids the DBA in identifying and, thus, responding to changes in requirements of the system and the designers who use the system.

SELECTED BIBLIOGRAPHY

A. References Cited

1. Eastman, Charles M. "Database Facilities for Engineering Design," Proceedings of the IEEE, Vol. 69, No. 10 (October 1981), pp. 1249-1263.

2. Lee, Benjamin and Casey Jones. "CAD Tools Must Change to Meet the Needs of VLSI," Electronics, November 17, 1981, pp. 108-117.

3. Reitmeyer, Randolph. "CAD for Military Systems, an Essential Link to LSI, VLSI and VHSI Technology," IEEE Proceedings of the 18th Design Automation Conference, pp. 3-12.

4. Myers, Ware. "CAD/CAM: The Need for a Broader Focus," Computer, January 1982, pp. 105-117.

5. Rosenberg, Lawrence. "The Evolution of Design Automation to Meet the Challenges of VLSI," IEEE Proceedings of 17th Design Automation Conference, pp. 3-10.

6. Panel Discussion. "Will Design Tools Catch Up to VLSI Design Needs?" IEEE Proceedings of 18th Design Automation Conference, pp. 544-556.

7. Raymond, T. "LSI/VLSI Design Automation," Computer, July 1981, pp. 89-101.

8. Panel Discussion. "Government Interest and Involvement in Design Automation Development," IEEE Proceedings of 18th Design Automation Conference, pp. 330-346.

9. Marshall, M. "VLSI Pushes Super-CAD Techniques," Electronics, July 31, 1980, pp. 73-80.

10. Eisenberg, H. "CADMON: Improving the CAD System Human Interface," IEEE Proceedings of 15th Design Automation Conference, pp. 353-462.

11. Satini, C. and M. Lenzerini. "INCOD: A System for Interactive Conceptual Data Base Design," IEEE Proceedings of 18th Design Automation Conference, pp. 546-554.

12. Newton, A. R. "Computer Aided Design of VLSI Circuits," Proceedings of the IEEE, Vol. 69, No. 10 (October 1981), pp. 1189-1199.

13. Fosdick, H. "Productivity in a Data Base Environment," Datamation, August 1982, pp. 109-112.

14. Yeh, Raymond, Agustin Araya, and Philip Chang. "Software and Data Base Engineering--Towards a Common Design Methodology," in Issues in Data Base Management, H. Weber and A. Wasserman, eds., pp. 109-123. New York: North Holland Publishing Company, 1979.

15. Sundgren, Dr. B. "Data Base Design in Theory and Practice," in Issues in Data Base Management, H. Weber and A. Wasserman, eds., pp. 3-37. New York: North Holland Publishing Company, 1979.

16. Wasserman, Anthony. "A Software Engineering View of Data Base Management," in Issues in Data Base Management, H. Weber and A. Wasserman, eds., pp. 41-63. New York: North Holland Publishing Company, 1979.

17. Weber, Herbert. "Modularity in Data Base System Design: A Software Engineering View of Data Base Systems," in Issues in Data Base Management, H. Weber and A. Wasserman, eds., pp. 65-91. New York: North Holland Publishing Company, 1979.

18. Earnest, Christopher. "Database Modularity," in Issues in Data Base Management, H. Weber and A. Wasserman, eds., pp. 93-97. New York: North Holland Publishing Company, 1979.

19. Panel's Papers on Wasserman and Weber's Articles, in Issues in Data Base Management, pp. 99-107. New York: North Holland Publishing Company, 1979.

20. Foulk, P. W., et al. "Aids--An Integrated Design System for Digital Hardware," IEEE Proceedings, Vol. 127, No. 2 (March 1980), pp. 45-57.

21. Nash, D. and H. Willman. "Software Engineering Applied to Computer-Aided Design (CAD) Software Development," Proceedings of the 18th Design Management Conference, pp. 530-539.

22. Korenjak, A. J. and A. H. Eger. "An Integrated CAD Data Base System," Proceedings of the 12th Design Automation Conference, pp. 399-406.

23. Breuer, M. A. and A. D. Friedman. "Initial Design Concepts for an Advanced Design Automation System," Proceedings of the 11th Design Automation Conference, pp. 366-371.

24. Smith, D. and B. Wagner. "A Low Cost, Transportable, Data Management System for LSI/VLSI Design," Proceedings of the 19th Design Automation Conference, pp. 283-290.

25. Martin, James. Computer Data-Base Organization. 2d ed. Englewood Cliffs, NJ: Prentice-Hall, 1977.

26. Wiederhold, G. Database Design. New York: Academic Press, 1977.

27. Date, C. J. An Introduction to Database Systems. 3d ed. Reading, Mass.: Addison-Wesley Publishing Company, 1981.

28. Ullman, J. D. Principles of Database Systems. Rockville, MD: Computer Science Press, 1980.

29. Weinberg, Victor. Structured Analysis. New York: Yourdon, Inc., 1980.

30. Carter, Harold A. "A Plan for Digital Systems Design Automation," Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1981.

31. Inmon, W. H. and L. J. Friedmon. Design Review Methodology for a Data Base Environment. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

32. Schaeffer, Howard. Data Center Operations. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

33. Tsichritzis, D. and F. Lochovsky. "Designing the Data Base," Datamation, August 1978, pp. 147-151.

34. Holland, Robert. "DBMS: Developing User View," Datamation, February 1980, pp. 141-144.

B. Related Sources

The following "uncited" bibliographic references provide relevant reading for the material in Chapters I and II. These references mainly cover the background areas of LSI and VLSI design.

Eastman, Charles. "Recent Developments in Representation in the Science of Design," IEEE Proceedings of the 18th Design Automation Conference, pp. 13-21.

Breuer, M. A., ed. Design Automation of Digital Systems--Theory and Techniques. Englewood Cliffs, NJ: Prentice-Hall, 1972.

Breuer, M. "Recent Developments in the Automated Design and Analysis of Digital Systems," Proceedings of the IEEE, Vol. 60, No. 1 (January 1972), pp. 12-23.

Breuer, M., A. Friedman, and A. Iosupovicz. "A Survey of the State of the Art of Design Automation," Computer, October 1981, pp. 58-75.

Sahni, S. and A. Bhatt. "The Complexity of Design Automation Problems," Proceedings of the 17th Design Automation Conference, pp. 402-410.

Cable, Hobart S. "Super-CAD: An Integrated Structure for Design Automation," Air Force Institute of Technology Thesis, Wright-Patterson AFB, Ohio, 1982.

Mead, C. and L. Conway. Introduction to VLSI Systems. Reading, Mass.: Addison-Wesley, 1980.

Seitz, Charles, ed. Proceedings of the Calteck Conference on Very Large Scale Integration, California Institute of Technology, 1979.

Soukup, Jiri. "Circuit Layout," Proceedings of the IEEE, Vol. 69, No. 10 (October 1981), pp. 1281-1304.

Barbe, D. F. "VHSIC Systems and Technology," Computer, February 1981, pp. 13-22.

van Cleemput, W. M. "An Hierarchical Language for the Structural Description of Digital Systems," Proceedings of the 14th Design Automation Conference, pp. 377-382.

Duley, J. and D. Dietmeyer. "A Digital System Design Language (DDL)," Vol. C-17, No. 9 (September 1968), pp. 850-861.

Monachino, M. "Design Verification System for LSI Designs," Proceedings of the 19th Design Automation Conference, pp. 83-95.

Kawato, N. and T. Saito. "Design and Verification of Large-Scale Computers by Using DDL," Proceedings of the 16th Design Automation Conference, pp. 360-366.

Prioste, J. "Macrocell Approach Customizes Fast VLSI," Electronic Design, June 7, 1980, pp. 159-167.

Singleton, P. and N. Croker. "Practical Automated Design of LSI for Large Computers," Proceedings of the 17th Design Automation Conference, pp. 556-559.

Yamada, A, et al. "Automatic Test Generation for Large Digital Circuits," Proceedings of the 14th Design Automation Conference, pp. 78-82.

McWilliams, T. "Verification of Timing Constraints on Large Digital Systems," Proceedings of the 17th Design Automation Conference, pp. 139-147.

Hickling, R. and G. Case. "Automating Test Generation Closes the Design Loop," Electronics, November 30, 1981, pp. 129-133.

Sasaki, T., et al. "MIXS: A Mixed Level Simulator for Large Digital System Logic Verification," Proceedings of the 17th Design Automation Conference, pp. 626-633.

Dunlop, A. "SLIM--The Translation of Symbolic Layouts into Mask Data," Proceedings of the 17th Design Automation Conference, pp. 595-602.

Freund, V., Jr. and J. Guerin. "Automated Conversion of Design Data for Building the IBM 3081," Proceedings of the 19th Design Automation Conference, pp. 96-103.

Chu, Y. "Computer System Design Description," Proceedings of the 19th Design Automation Conference, pp. 842-844.

Basset, P. and G. Saucier. "Top Down Design and Testability of VLSI Circuits," Proceedings of the 19th Design Automation Conference, pp. 851-857.

Adshead, H. G. "Towards VLSI Complexity: The DA Algorithm Scaling Problem: Can Special Design Automation Hardware Help," Proceedings of the 19th Design Automation Conference, pp. 339-343.

Leyking, L. W. "Data Base Considerations for VLSI," Proceedings of the Cal Tech Conference on VLSI, California Institute of Technology, 1979.

"User's Guide for the ERADCOM MP2D Program."

Vladimirescu, A., A. R. Newton, and D. O. Pederson. "SPICE Version 2G.0 User's Guide." Department of Electrical Engineering and Computer Science, University of California, Berkeley, 22 September 1980.

"Scoap User's Guide," Lawrence Goldstein, Division 2113, Sandia Laboratories, Albuquerque, NM 87185.

Barbacci, Mario and Andrew Nagle. The Symbolic Manipulation of Computer Descriptions--an ISPS Simulator. Departments of Computer Science and Electrical Engineering, Carnegie-Mellon University, April 1979.

"PCPRA User's Manual--Volume 5," Westinghouse Electric Corporation, Baltimore, MD, April 1977.

"Logic 4 User's Manual--Volume 3," Westinghouse Electric Corporation, Baltimore, MD, April 1977.

"IC Layout--User's Manual--Volume 7," Westinghouse Electric Corporation, Baltimore, MD, April 1977.

Acken, John and Jerry Stauffer. "SALOGS User's Guide," Sandia Laboratories, Albuquerque, NM 87185.

The following "uncited" bibliographic references provide relevant reading for the material in Chapter IV, "Design of the DA System Model." These references discuss different design systems.

Nieng, K. Y. and Dennis Beckley. "Component Library for and Integrated DA System," IEEE Proceedings of 16th Design Automation Conference, pp. 437-44.

Eastman, C. "System Facilities for CAD Databases," ICCC Proceedings of 17th Design Automation Conference, pp. 50-56.

Zimmerman, G. "The Mimola Design System: A Computer-Aided Digital Processor Design Method," IEEE Proceedings of 17th Design Automation Conference, pp. 53-63.

Bering, D. "The Electronics Engineers Design Station," IEEE Proceedings of 17th Design Automation Conference, pp. 422-427.

Parker, A., D. Thomas, D. Siewiorek, M. Barbacci, et al. "The CMU Design Automation System," IEEE Proceedings of 16th Design Automation Conference, pp. 73-80.

David, B. T. "An Integrated CAD System for Architecture," IEEE Proceedings of 17th Design Automation Conference, pp. 218-225.

Sanders, J. M. Jenkins, et al. "PHILO--A VLSI Design System," IEEE Proceedings of 19th Design Automation Conference,

Sanborn, Jere. "Evolution of the Engineering Design System Data Base," IEEE Proceedings of 19th Design Automation Conference, pp. 214-218.

Friedenson, R. "Dsigner's Workbench: Delivery of CAD Tools," IEEE Proceedings of the 19th Design Automation Conference, pp. 15-22.

Oakes, M. F. "The Complete VLSI Design System," IEEE Proceedings of 16th Design Automation Conference, pp. 452-460.

Brinton, J. "CHAS Seeks Title of Global CAD System," Electronics, February 10, 1981, pp. 100-104.

Lucke, V. H. "Recent CAD/CAM Development in General Electric," IEEE Proceedings of Systems/Man/Cybernetics Conference, pp. 871-874.

Wozny, M. "The Role of CAD/CAM in Universities," IEEE Proceedings of Systems/Man/Cybernetics Conference, pp. 886-870.

The following "uncited" bibliographic references provide relevant reading for the material in Chapter V, "Design of the Canonical Schema." These references discuss various topics concerning data bases, especially those within a design environment.

Kawano, I. "The Design of a Data Base Organization for an Electronic Equipment DA System," IEEE Proceedings of 15th Design Automation Conference, pp. 167-175.

Nash, Dan. "Topics in Design Automation Data Base," IEEE Proceedings of 15th Design Automation Conference, pp. 463-473.

Nash, D., P. Ciampi, and A. Donovan. "Control and Integration of a CAD Data Base," IEEE Proceedings of 13th Design Automation Conference, pp. 285-289.

"Concepts in CAD Data Base Structures," IEEE Proceedings of 13th Design Automation Conference, pp. 290-294.

Roberts, K., T. Baker, and D. Jerome. "A Vertically Organized Computer Aided Design Data Base," IEEE Proceedings of 18th Design Automation Conference, pp. 595-602.

Foster, J. C. "The Evolution of an Integrated Data Base," IEEE Proceedings of 12th Design Automation Conference, pp. 394-398.

Sucher, D. and D. Wann. "A Design Aids Data Base for Digital Components," IEEE Proceedings of 16th Design Automation Conference, pp. 414-420.

Wiederhold, G., et al. "A Data Base Approach to Communication in VLSI Design," Stanford Department of Computer Science, Report No. STAN-CS-80-826, October 1980.

Wong, S. and W. A. Bristol. "A Computer-Aided Design Data Base," IEEE Proceedings of 16th Design Automation Conference.

Bennett, J. "A DB MS for Design Engineers," IEEE Proceedings of Design Automation Conference, pp. 268-273.

O'Neill, L. "A Retrospective on SW Engineerig in Design Automation," IEEE Proceedings of 19th Design Automation Conference, pp. 10-14.

Katz, Randy. "A Data Base Approach for Managing VLSI Design Data," IEEE Proceedings of 19th Design Automation Conference, pp. 274-282.

Hubbard, George. Computer-Assisted Data Base Design. New York: Van Nostrand Reinhold Company, 1981.

Codd, E. F. "Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol. 13, No. 6 (June 1970), pp. 377-387.

NG, Peter. "Further Analysis of the Entity-Relationship Approach to Data Base Design," IEEE Transactions on Software Engineering, Vol. SE-7, No. 1 (January 1981), pp. 85-99.

Chang, S. K. and W. H. Cheng. "A Methodology for Structured Database Decomposition," IEEE Transactions on Software Engineering, Vol. SE-6, No 2 (March 1980), pp. 205-218.

Teorey, T. and J. Fry. "The Logical Record Access Approach to Data Base Design," Computing Survey, Vol. 12, No. 2 (June 1980), pp. 179-211.

Frasson, C. "Generalized Translation in a Data Base System," IEEE Proceedings of 15th Design Automation Conference, pp. 176-181.

Hoskins, E. "Descriptive Data Bases in Some Design/ Manufacturing Environments," IEEE Proceedings of 16th Design Automation Conference, pp. 421-436.

Reside, K. and T. Seiter. "The Evolution of an Integrated Data Base," Datamation, September 1974.

Wilmore, James. "The Design of an Efficient Data Base to Support an Interactive LSI Layout System," IEEE Proceedings of 16th Design Automation Conference, pp. 445-451.

Chu, W. and P. Chen, eds. "Entity-Relationship Model and Data Systems," Chapter 4 in Tutorial: Centralized and Distributed Data Base Systems, pp. 157-165. IEEE Computer Society.

The following "uncited" bibliographic references provide relevant reading for the material in Chapter VII, "Recommendations." These references discuss various topics concerning data base models and DBA functions and tools.

Rasdolf, W. and A. Kutay. "Maintenance of Integrity During Concurrent Access in a Building Design Data Base," Computer Aided Design, Vol. 14, No. 4 (July 1982), pp. 201-207.

Inmon, W. and L. Friedman. Design Review Methodology for a Data Base Environment. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Richardson, F. "Important Criteria in Selecting Engineering Work Stations," IEEE Proceedings of 19th Design Automation Conference, pp. 440-444.

Noon, W., K. Robbins, and M. Roberts. "A Design System Approach to Data Integrity," IEEE Proceedings of 19th Design Automation Conference, pp. 699-705.

Sparr, T. "A Language for a Scientific and Engineering Data Base System," IEEE Proceedings of 19th Design Automation Conference, pp. 865-871.

Howden, William. "A Survey of Static Analysis Methods," IEEE Tutorial: Software Testing and Validation Techniques, 1978, pp. 83-96.

Jensen, Randall. "Tutorial: Structured Programming," Computer, Vol. 14, No. 3 (March 1981), pp. 31-78.

Zelkowitz, M. "Perspectives on Software Engineering," Computing Surveys, Vol. 10, No. 2 (June 1978), pp. 197-216.

Ross, D. "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1 (January 1977), pp. 16-34.

Stevens, W. P., G. Meyers, and L. L. Constantine. "Structured Design," IBM Systems Journal, Vol. 13, No. 2 (1974).

Morgan, D. E. and D. J. Taylor. "A Survey of Methods of Achieving Reliable Software," Computer, February 1977, pp. 44-56.

Deutsch, M. "Software Project Verification and Validation," Computer, April 1981, pp. 54-70.

Barnhardt, Robert. "Implementing Relational Data Base," Datamation, October 1980, pp. 161-172.

Haynie, Mark. "The Relational/Network Hybrid Data Model for Design Automation Data Bases," IEEE Proceedings of 18th Design Automation Conference, pp. 646-652.

Michaels, A. B., B. Mittmann, and R. Carlson. "A Comparison of the Relational and CODASYL Approaches to Data Base Management," Computing Survey, Vol. 8, No. 1 (March 1976), pp. 125-151.

"Relational DB: A Practical Foundation for Productivity," Communications of the ACM, Vol. 25, No. 2 (February 1982), pp. 109-117.

Emich, H. "Design Data Base Configuration Control," IEEE Proceedings of 10th Design Automation Conference, pp. 274-281.

Lozinskii, E. L. "Performance Consideration in Relational Data Base Design: Improving Performance," in Databases: Improving Usability and Responsiveness, pp. 273-294. New York: Academic Press, Inc., 1978.

Sidle, Thomas. "Weaknesses of Commercial DBMSs in Engineering Applications," IEEE Proceedings of 17th Design Automation Conference, pp. 57-61.

APPENDICES

Appendix A

Design Task Data Diagrams

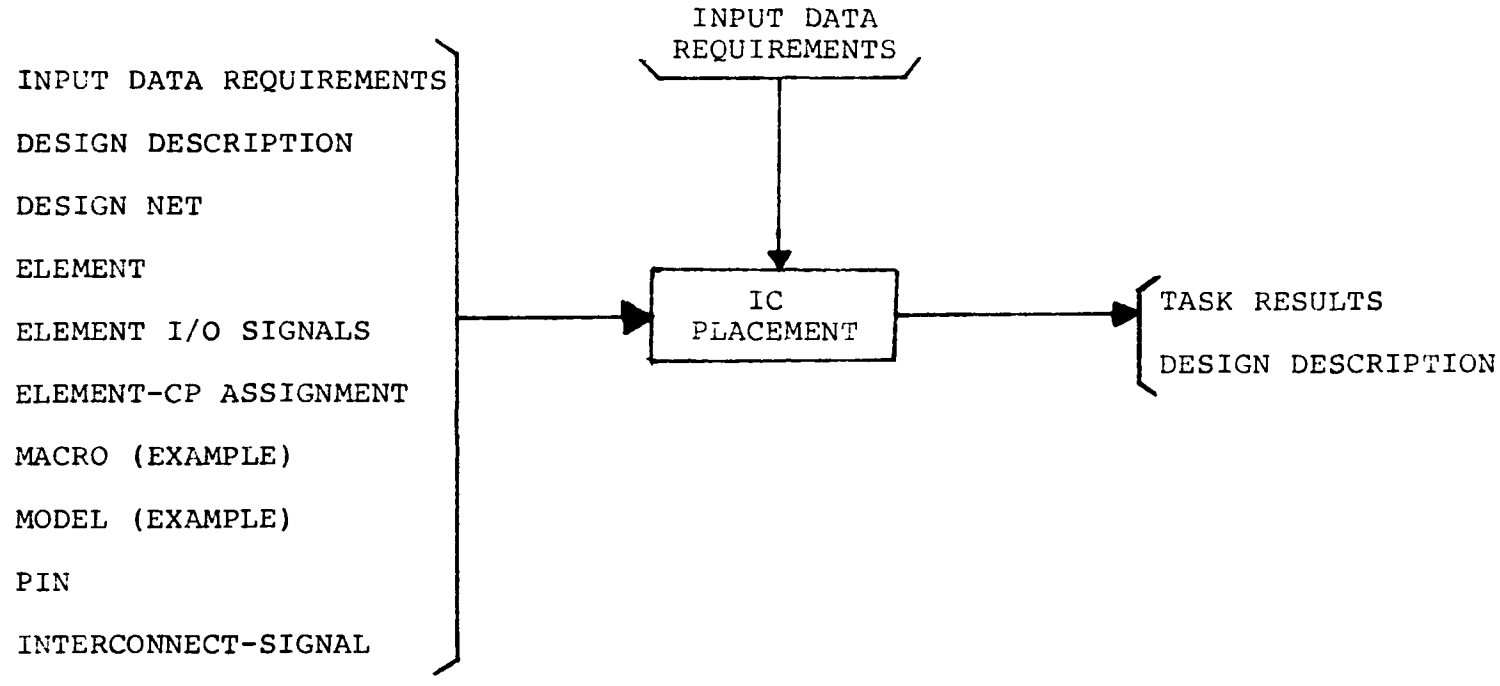
The following Design Task Data Diagrams describe design task-views. The descriptive data items used in these diagrams are attributes of a Relation's Details. The data is grouped into three categories:

INPUT (left); CONTROL (upper); OUTPUT (right).

The design task is labeled in the middle box.

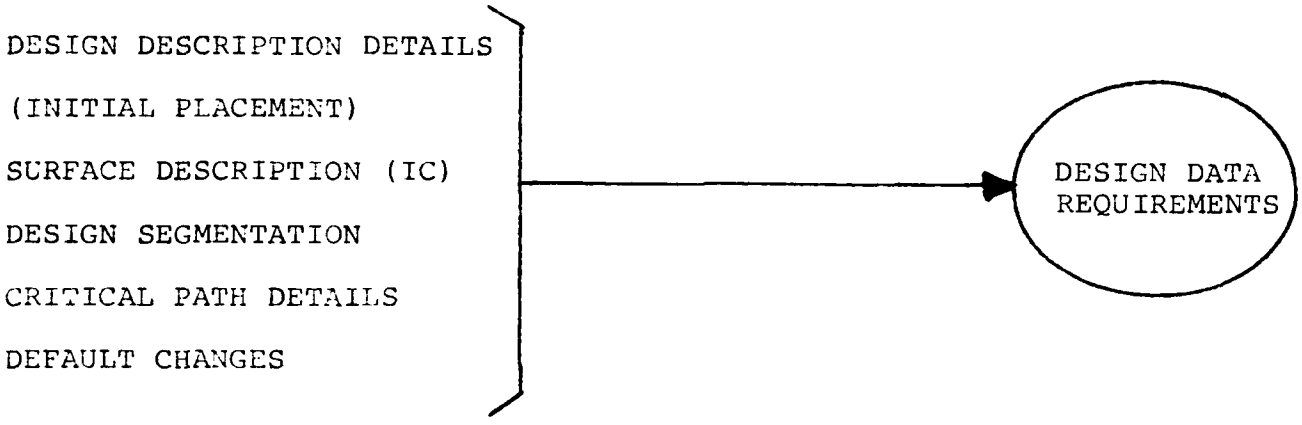
As previously mentioned, because of the generic data concept, many of the design tasks require much of the same data. However, a closer examination showing the specific data required by a design task, reveals the unique specific data requirements of each task. The detailed data descriptions are expansions of some of the relations which provide good examples.

The design tasks included are the major, frequently used design tasks. They are provided to aid the reader in understanding the data requirements of the tasks, the specific data, the generic data, and the relationship of the specific and generic data. It also shows the specification of the generic data into Relations, which are the building blocks of the data organization of the IDDB.



GENERIC DATA DIAGRAM

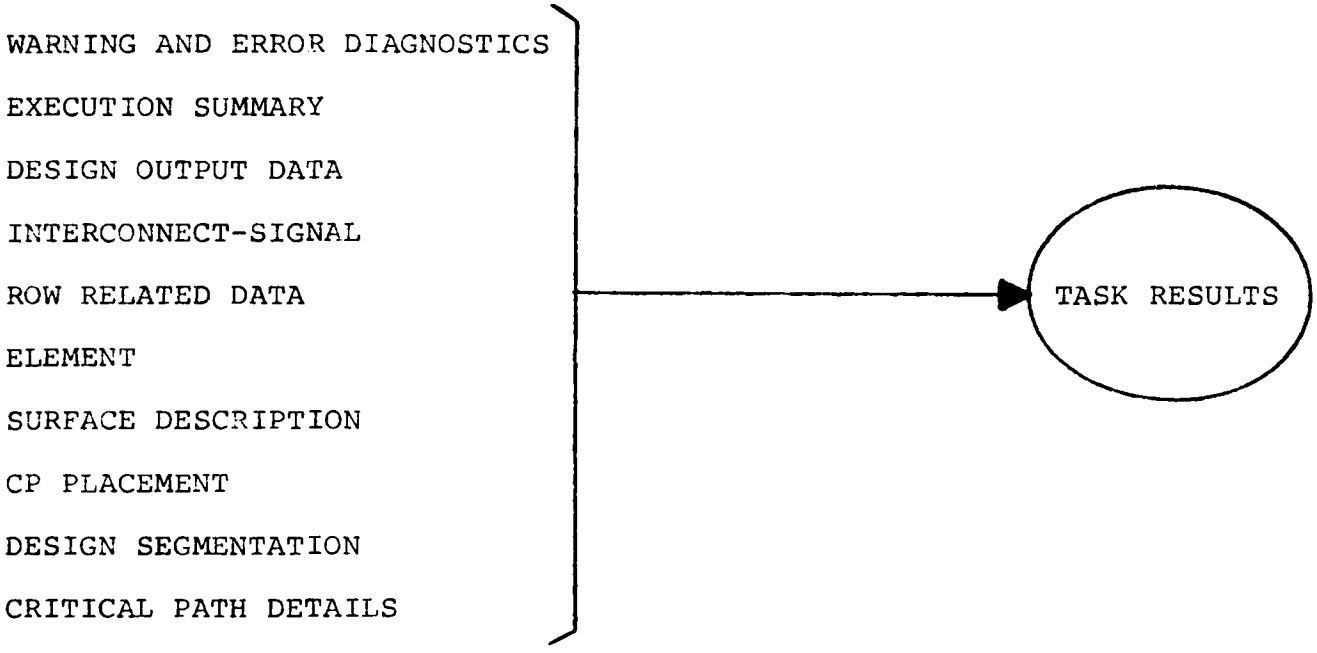
IC PLACEMENT



SPECIFIC DATA DIAGRAM

IC PLACEMENT

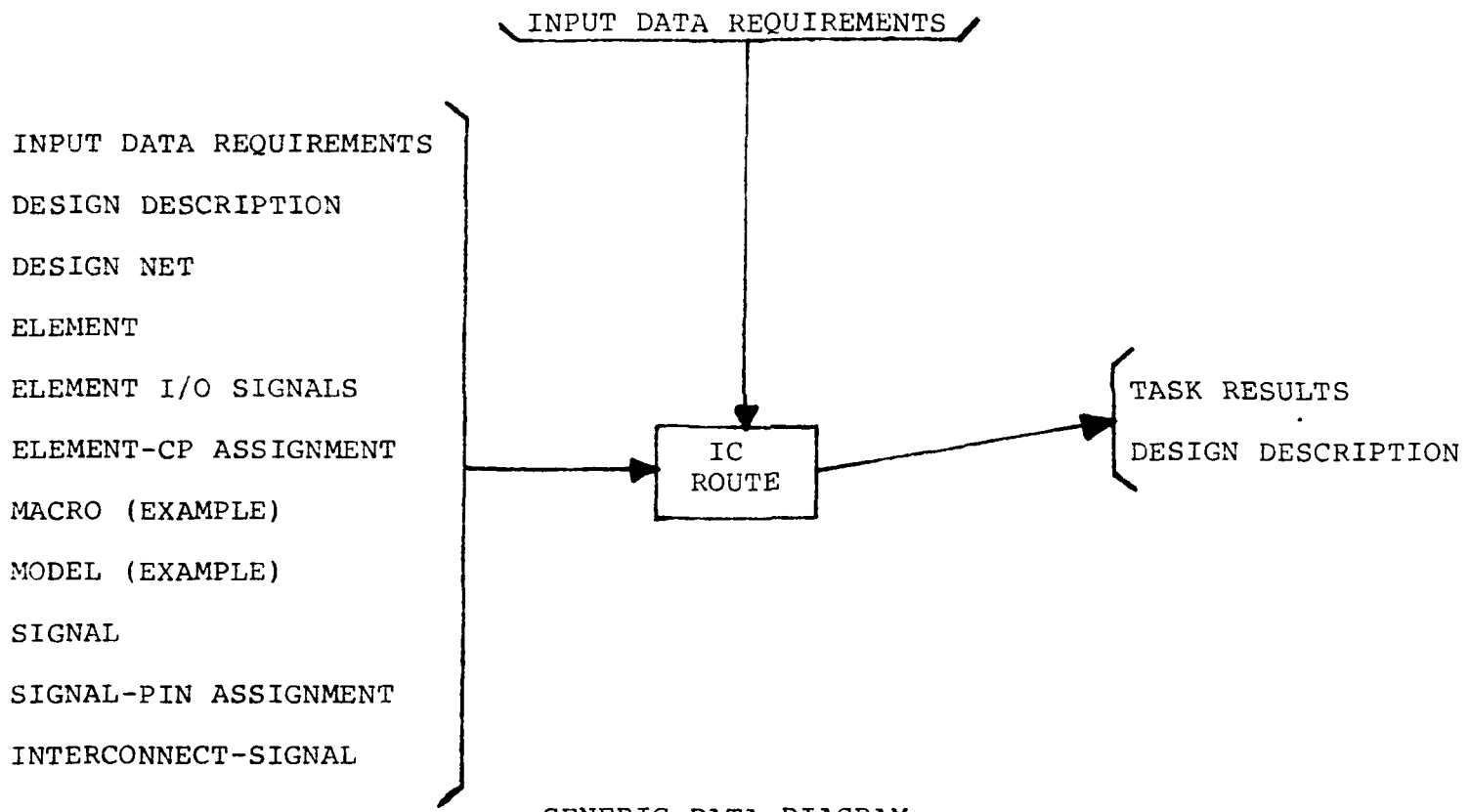
DESIGN DATA REQUIREMENTS



SPECIFIC DATA DIAGRAM

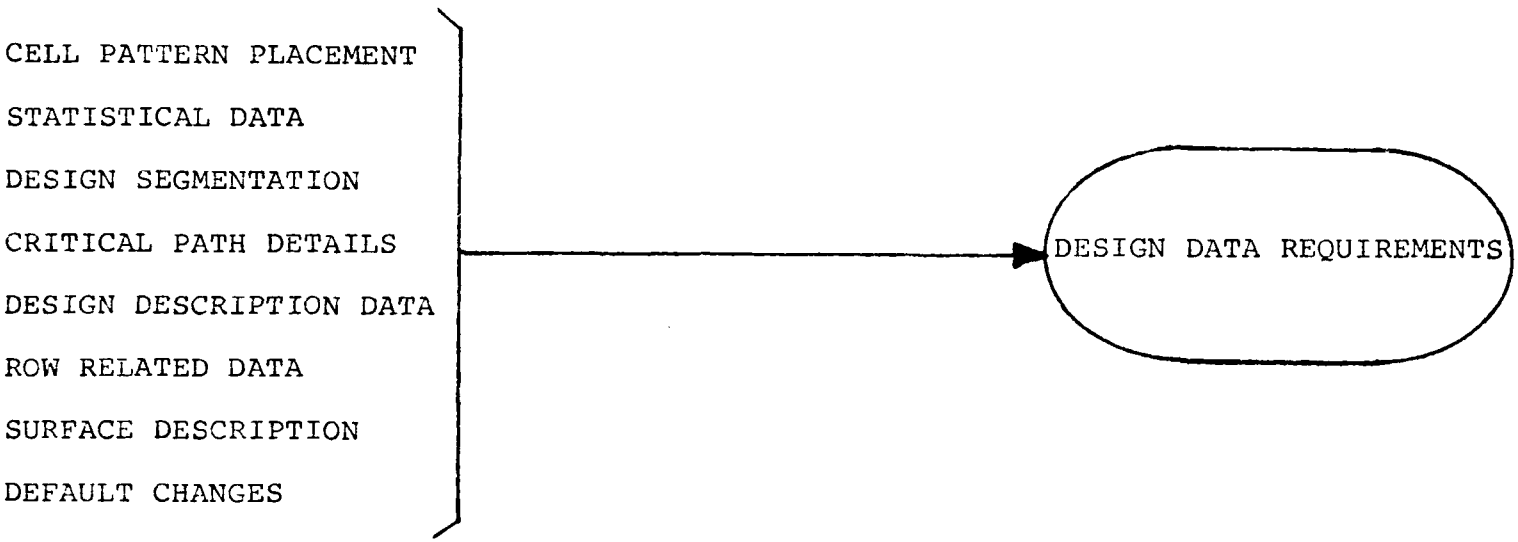
IC PLACEMENT

TASK RESULTS



GENERIC DATA DIAGRAM

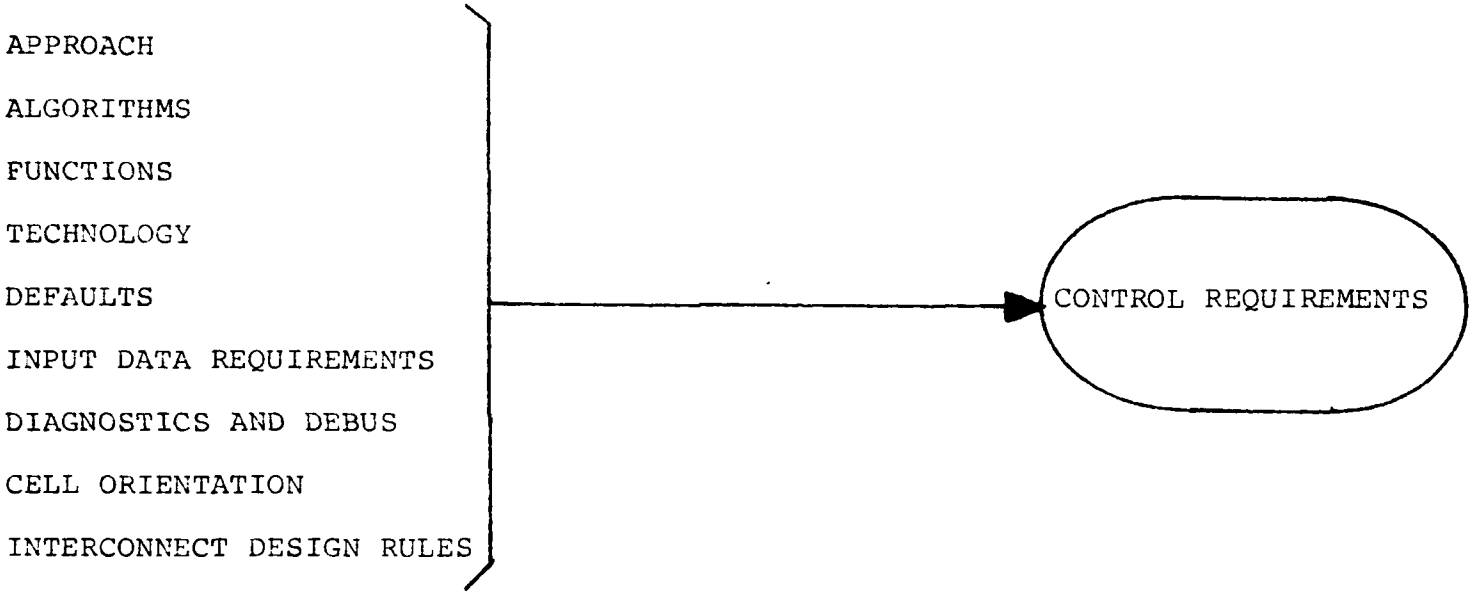
IC ROUTE



SPECIFIC DATA DIAGRAM

IC ROUTE

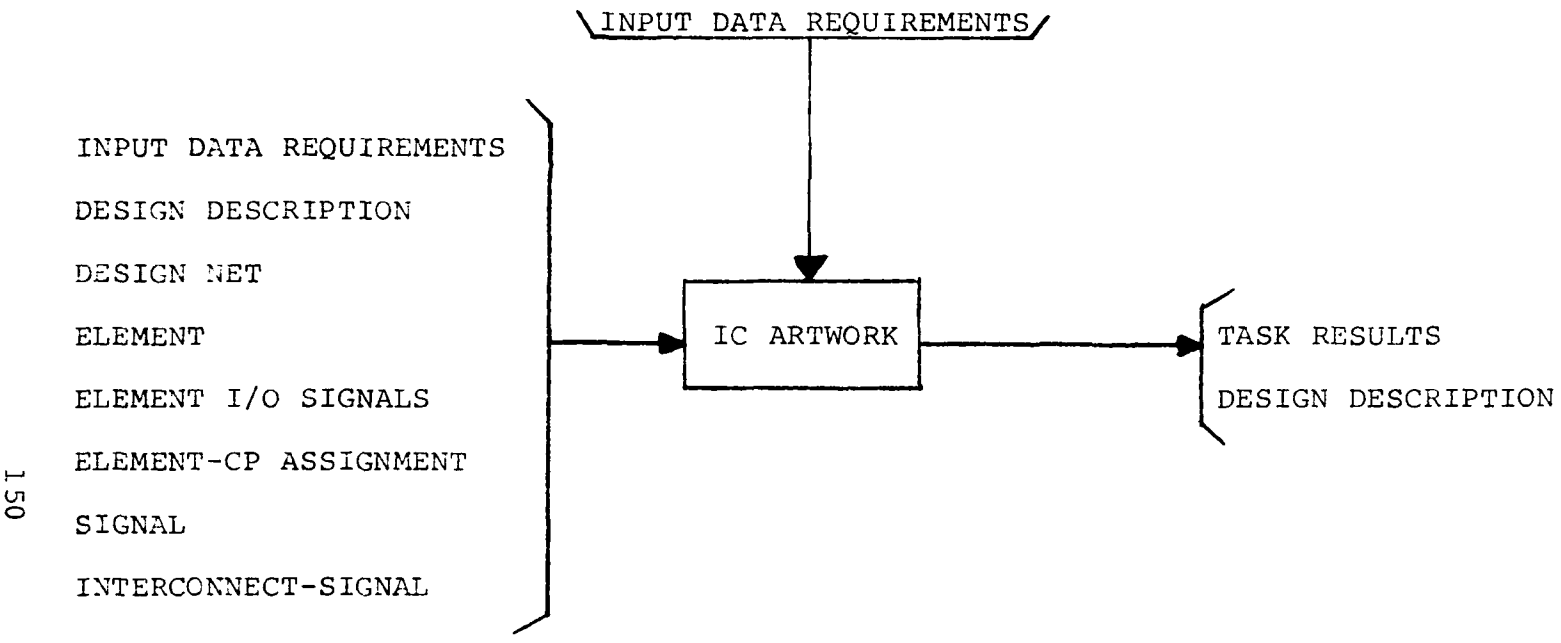
DESIGN DATA REQUIREMENTS



SPECIFIC DATA DIAGRAM

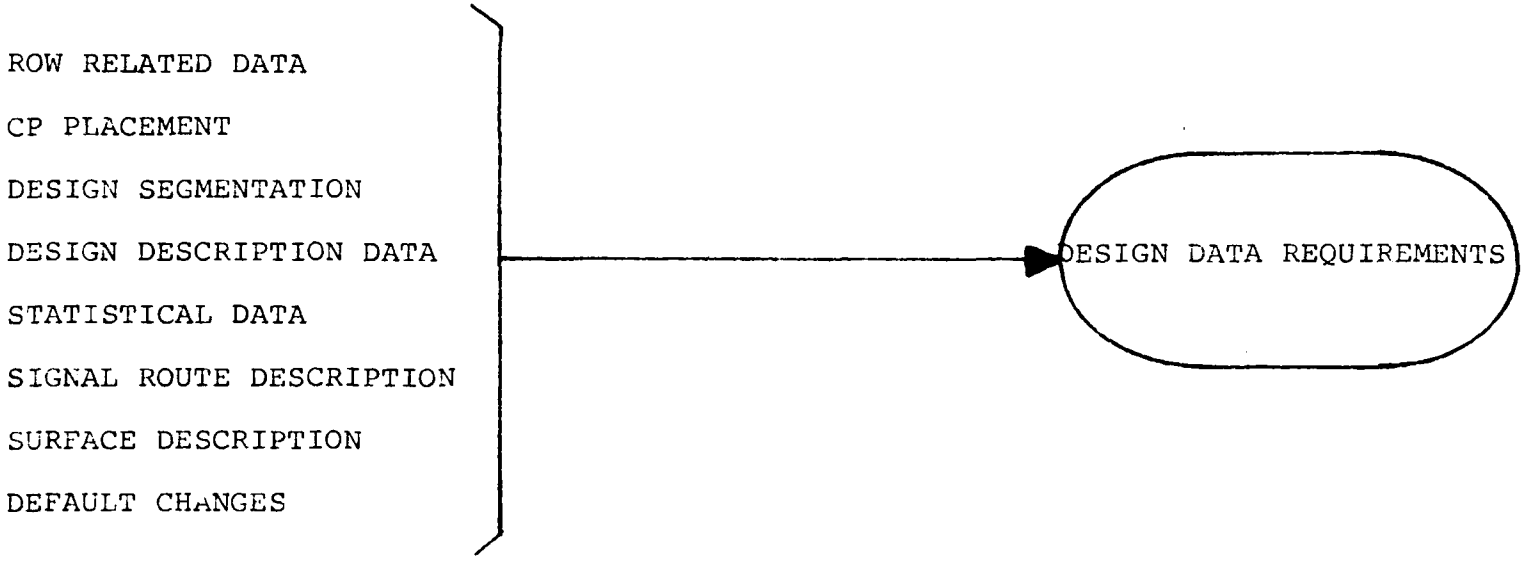
IC ROUTE

CONTROL REQUIREMENTS



GENERIC DATA DIAGRAM

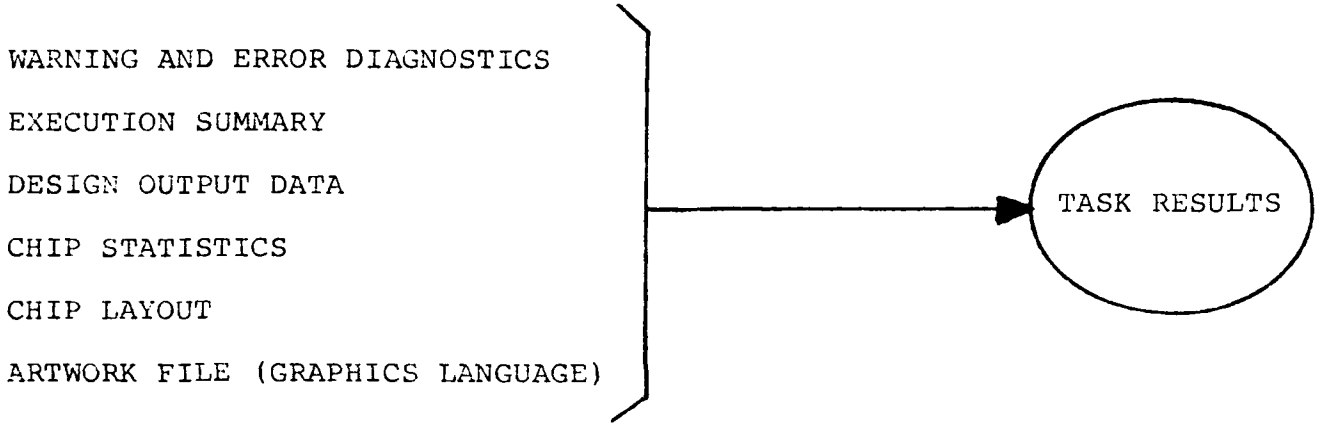
IC ARTWORK



SPECIFIC DATA DIAGRAM

IC ARTWORK

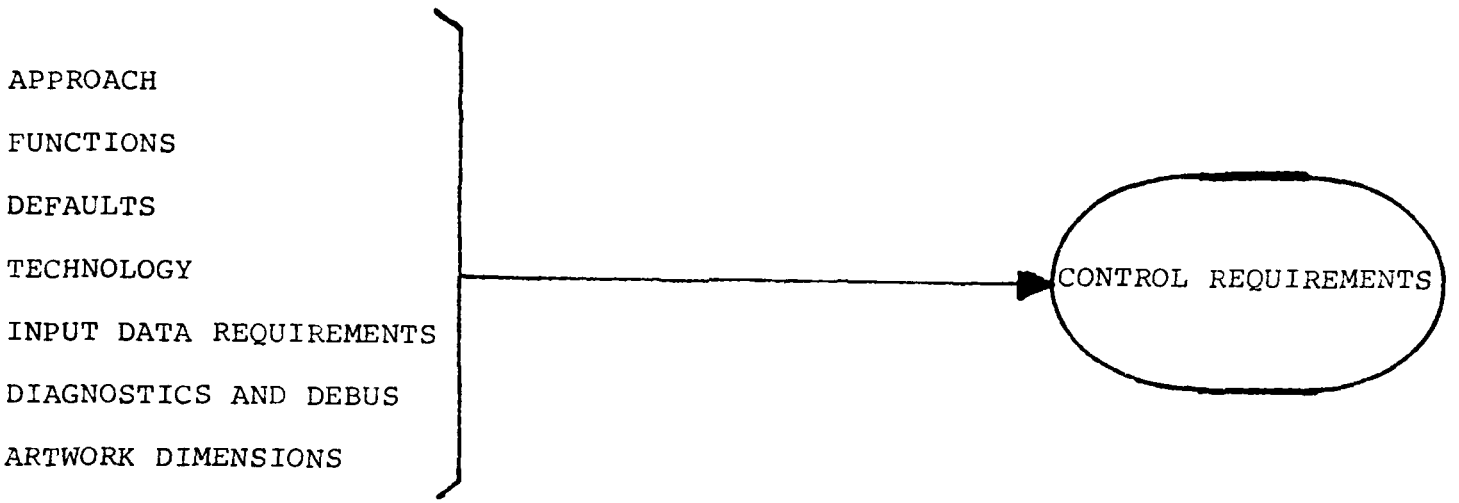
DESIGN DATA REQUIREMENTS



SPECIFIC DATA DIAGRAM

IC ROUTE

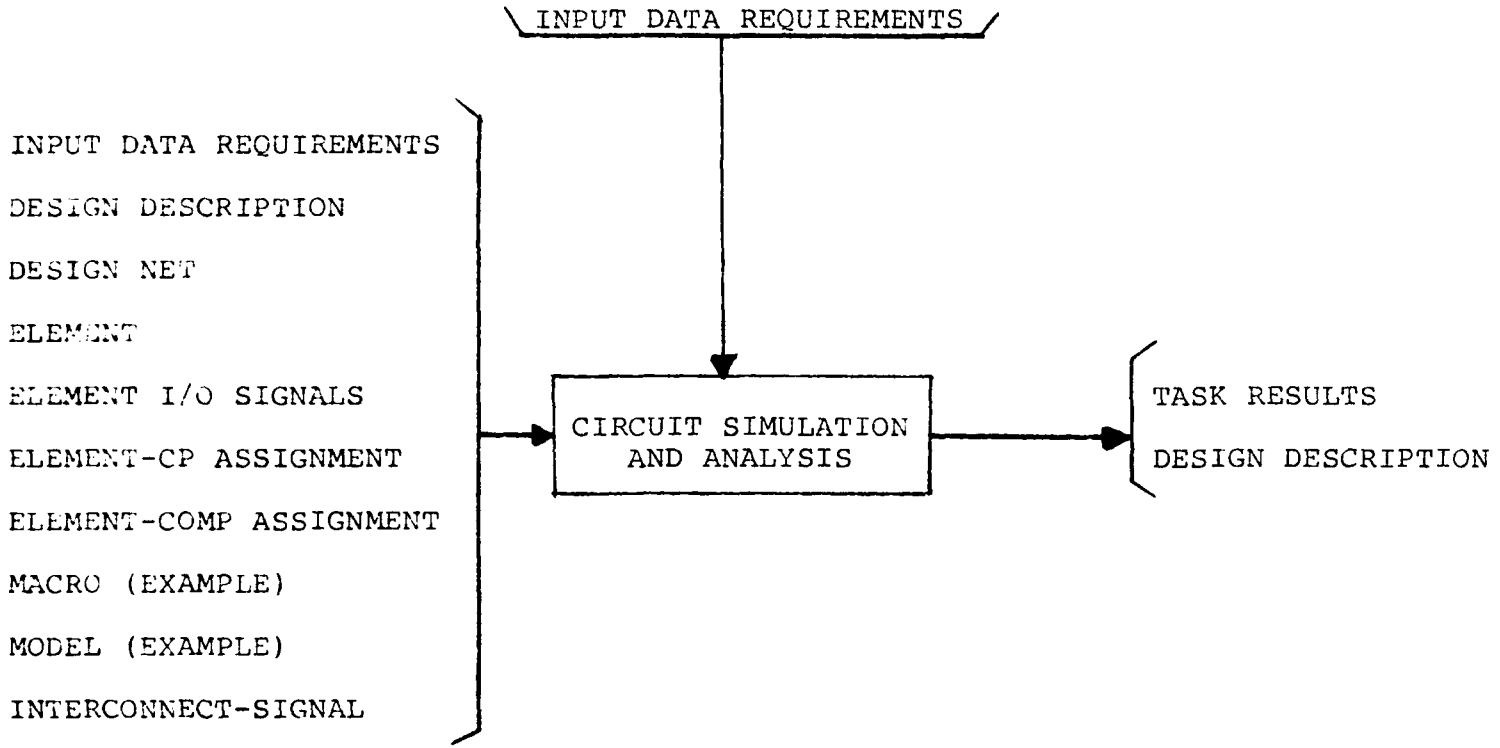
TASK RESULTS



SPECIFIC DATA DIAGRAM

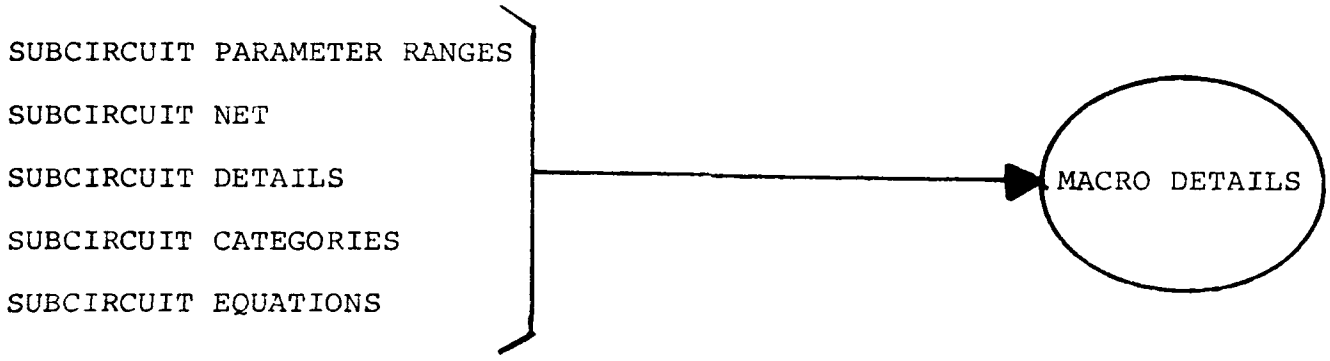
IC ARTWORK

CONTROL REQUIREMENTS



GENERIC DATA DIAGRAM

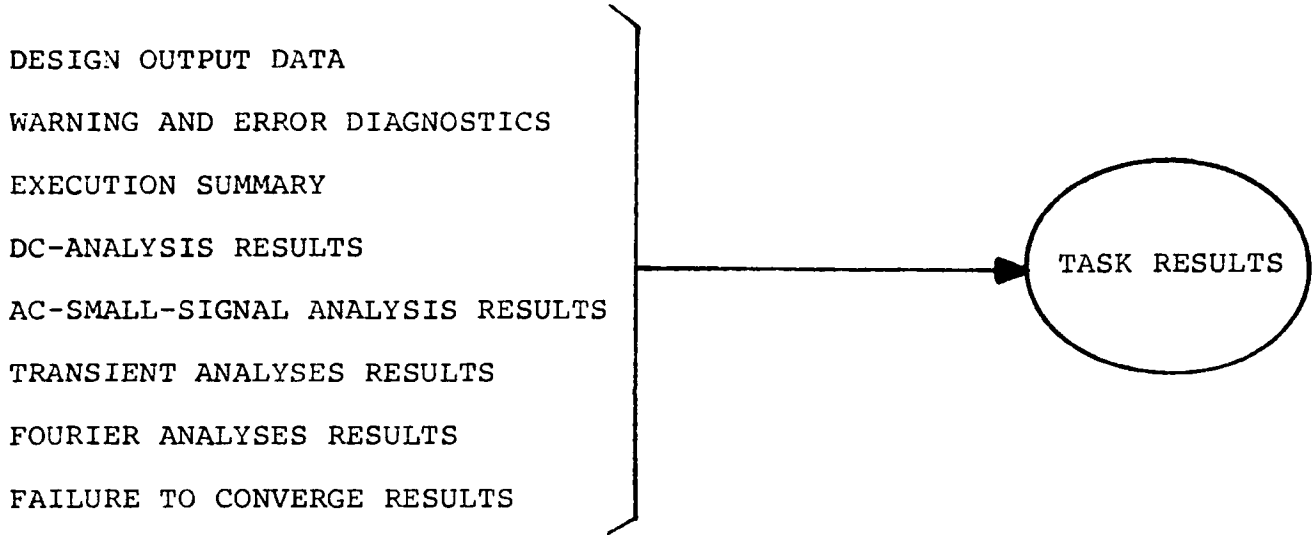
CIRCUIT SIMULATION AND ANALYSIS



SPECIFIC DATA REQUIREMENTS

CIRCUIT SIMULATION AND ANALYSIS

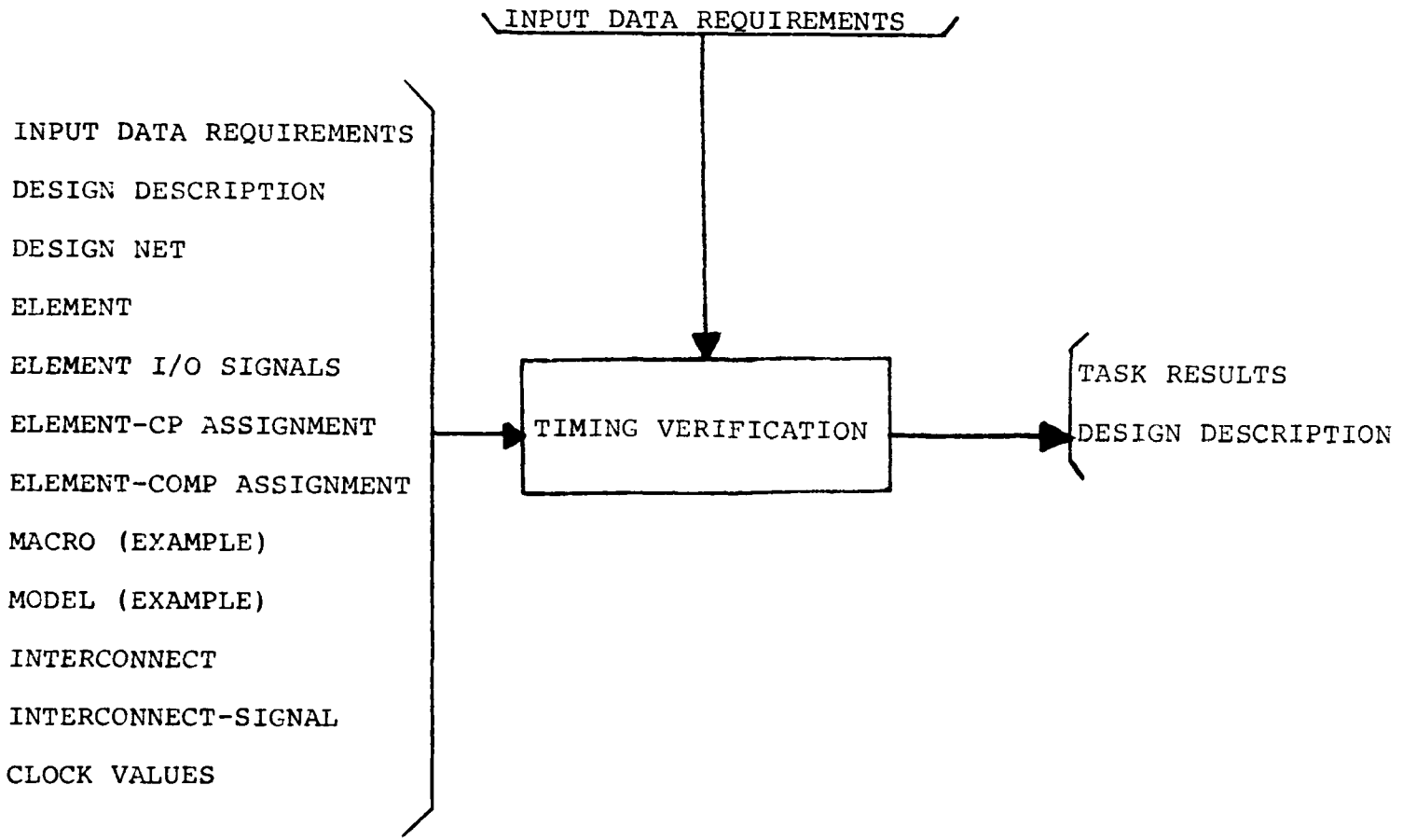
MACRO DETAILS



SPECIFIC DATA REQUIREMENTS

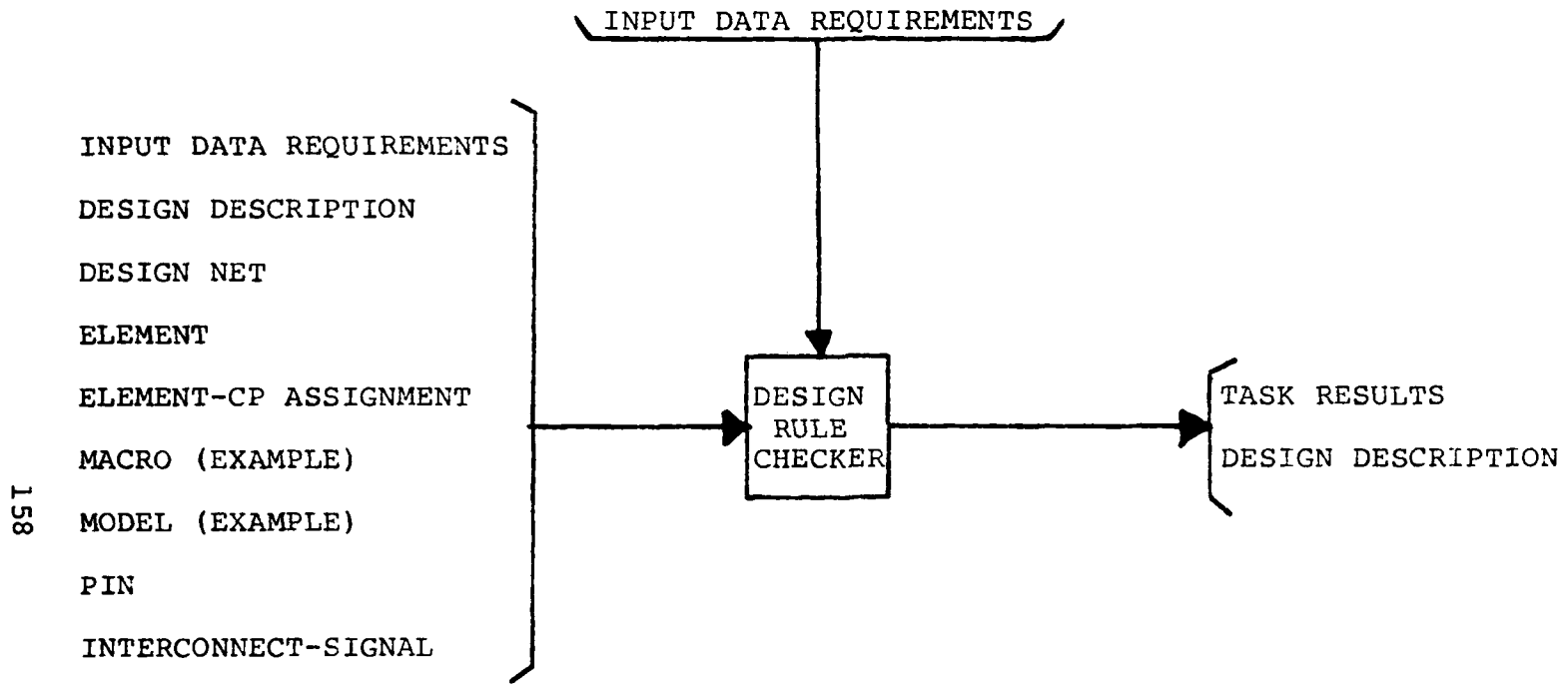
CIRCUIT SIMULATION AND ANALYSIS

TASK RESULTS



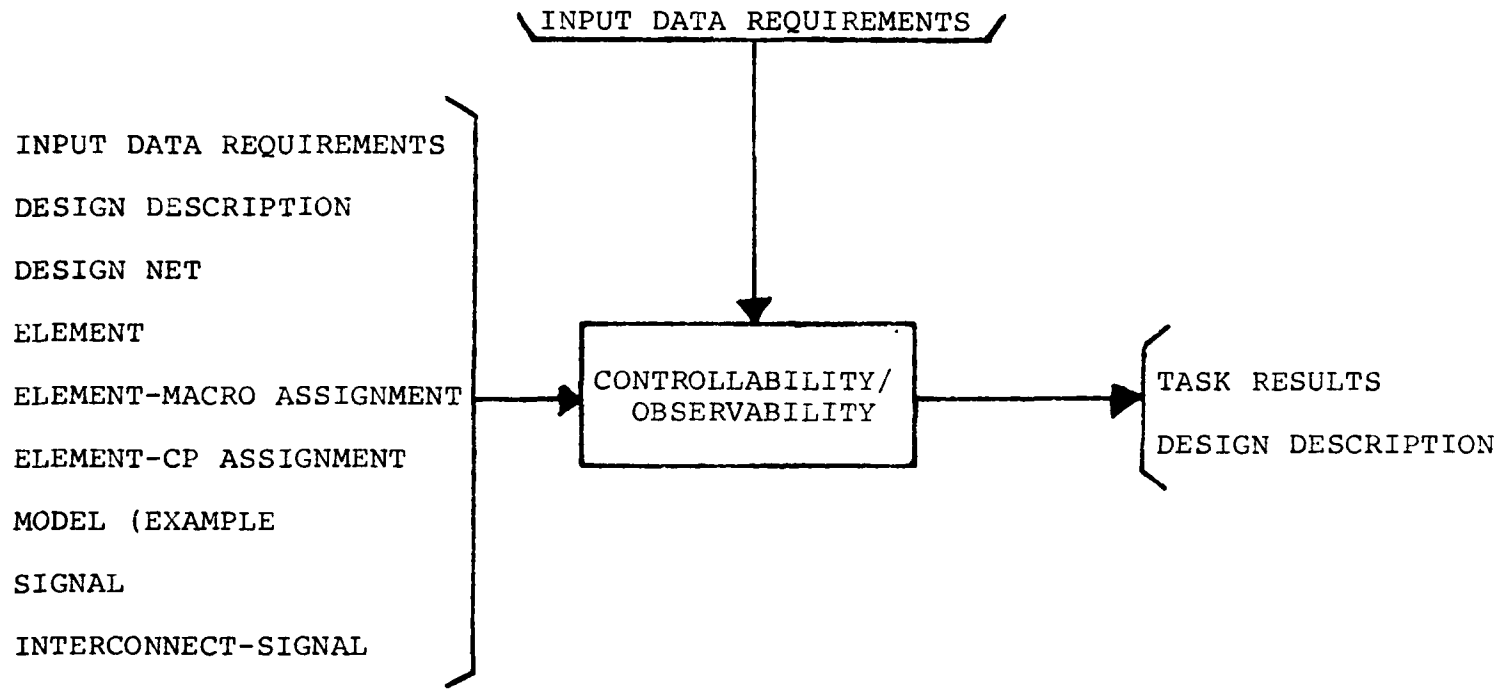
GENERIC DATA DIAGRAM

TIMING VERIFICATION



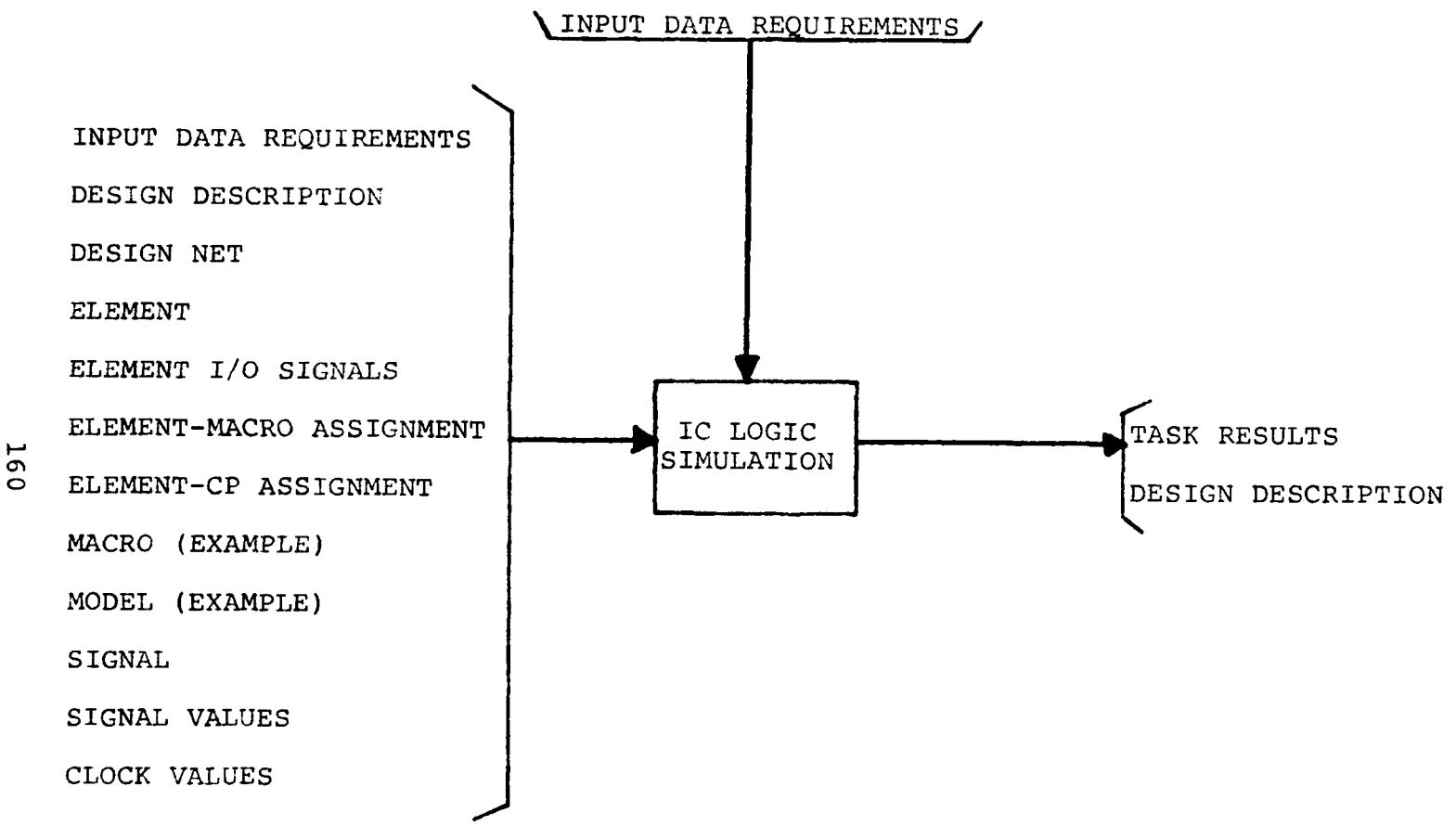
GENERIC DATA DIAGRAM

DESIGN RULE CHECKER



GENERIC DATA DIAGRAM

CONTROLLABILITY/OBSERVABILITY



GENERIC DATA DIAGRAM

IC LOGIC SIMULATION

Appendix B

Glossary of Terms

Data Base--a collection of multiple record types, containing the relationships between records, data aggregates, and data items (24:13). Another definition of data bases states that it is a collection of stored operational data used by the application systems of some particular enterprise (26:4). Since, in most systems, the term data base does not refer to all the record types, but to a specified collection of them. There can be several data bases in a system; however, the contents of each data base are considered disjoint.

Data Base System--a collection of data bases. There are three ways that data is organized in a data base: external, global logical data, and physical storage.

External Organization--concerned with the application programmer's view of the data. The programmer's view of the data is inherently defined by the application program being used.

Global Logical Base Organization--the overall organization or conceptual model for the data base from which multiple external organizations may be derived. It is the logical view of the data, entirely independent of the physical storage organization. It will be described in a data definition language which is part of the DBMS.

Physical Storage Organization--concerned with the physical representation, layout, and organization of the data on the storage units. It is concerned with the indices, pointers, chains, and other means of physically locating records, overflow areas, and how data operations (insert, delete, etc.) are performed on the storage medium.

Defaults--data categories that belong to the Project Independent data area. Each Application Program has default parameter values or Defaults.

Default Changes--data categories that belong to the Project Dependent data area. The user may make changes to the Default values; these changes are Default Changes.

Key--a key has two properties: (1) it uniquely identifies a data item or tuple, and (2) it is nonredundant. This means that no prime attribute can be discarded without destroying the uniqueness property of 1.

Prime Attribute--an attribute that is a member of several other prime attributes that constitutes the key.

Non-Prime Attribute--an attribute that is not a member of a key.

Schema--the overall logical data base description is often referred to as the schema, an overall model of the data, a conceptual model, or a conceptual schema. The conceptual model, then, is a view of the total data base content, and the conceptual schema is a definition of this view. The schema is a chart of the types of the data used, the names of the entities and attributes, and their relationships. Thus, the schema can be viewed as a skeleton upon which the data from the enterprise is attached.

Sub-Schema--simply one application programmer's view (i.e., an application program's data "expectation"). Many different sub-schemas can be derived from one schema. Two reasons for the sub-schema are to help avoid data complexity, where possible, and to aid in data security.

Task--a group of small related processes to be performed by Application Programs. Also a Design Task.

Task-View--the data requirements of a task. These requirements are the data, data format, and data type as required for specific Application Programs. It can also be a higher level view of the generic data requirements of a design task. These two levels of views are seen in the Design Task Data Diagrams of Appendix A.

Appendix C

Questionnaire

This questionnaire was informally used by the author to gather the data requirements of the DA System. There are two categories of questions:

1. Data Organization, and
2. Data Processing Requirements.

The first category identifies the data and the functional dependencies of the data. The second category identifies what transformations that the data undergoes and other data operations (33:147-151; 34:141-144).

QUESTIONNAIRE

A. Data Organization

1. What are the entities (data elements) of interest in each application program? Names?
2. What facts (Attributes) for each entity are relevant? Names?
3. What is the range of values for each attribute?
4. What are known dependencies between attributes of each identity?
5. What are unique identifiers (Keys) for each entity (if any)?
6. What are important relationships between entities?
7. What is the mapping property of each relationship? (1:1, 1:N, N:M)
8. What is the meaning and implication of each relationship?
9. What are the possible relationships, not used, but still meaningful?
10. What combination of relationships make sense as separate, identifiable relationships?

B. Data Processing Requirements

1. What transactions are required by each application program? (What are the overall and segmented operations done by the program?)
2. What kind of data access is required by each transaction?
3. At what frequency is the operation done?

4. What entities, attributes, and relationships are involved in each operation?

5. Is the processing of the operations priorized and/or are some operations done conditionally?

6. Is there an important prerequisite sequence for any transaction? Are they sequential or independent (can be run simultaneously)? What is the impact of the iterations, what data is affected, which transactions are affected?

7. How often are transactions done during a design cycle?

8. What is the output of each transaction; not the format, but changed entities, attributes, and relationships (include error codes).

9. What data, that has been retrieved or changed, must be saved in a work area for the designer?

10. What type of input is required for the transaction? (interactive, file, graphics, library (static), or direct input from another transaction).

11. List FUTURE transactions and/or functions that may be integrated into the system. Also, list which entities and relations will be used.

Appendix D

Software Engineering Tools and Techniques

1. Requirements

Requirements analysis, definition, and specification--understanding the problem precisely, developing unambiguous statements of system functions, decomposing the problem into manageable subparts.

Management of software development--
Staff organization, budgeting, planning, system integration, personnel deployment.

2. Design

Software design--
Use of well-defined methods for establishing the logical structure of a software system, creation of software "blueprint" and "breadboards."

examples:

- Structured Design (Yourdon-Constantine)
- Jackson Method
- Wernier-Orr Method
- SADT (SofTech)

- HIPO Charts
- Program Design Language(s)
- Decision Tables
- Flow Charts
- Design Walkthroughs

3. Implementation

Systematic Programming Methodology--
Techniques for reliably producing programs that are correct, including stepwise refinement and structured programming.

Programming Tools and Environments--
Text editors, debugging tools, operating systems, programming languages that support the programming process, portability, standards enforcement, top-down implementation, code walkthroughs, and performance analyzers.

4. Testing

Program Testing and Verification--
Construction of test cases to determine program correctness and performance characteristics, techniques for selecting test data, formal mathematical proof that a program meets stated specifications.

examples:

- Static Analyzers
- Dynamic Analyzers
- Execution Analyzers
- Test Data Generators
- Assertion Checkers
- Test Coverage Analyzers

Software Performance--

Analysis of algorithms, prediction, evaluation, and improvement of performance.

5. Maintenance

Viability--

System fit to requirements, adaptability, maintainability.

examples:

- Flow Chart Generators
- Data Dictionary
- Source Code Control
- Interface Analyzer/Checker

6. Documentation

Formal and informal requirements definitions and specifications, design representations, user manuals,

program descriptions, program readability and commenting practices.

examples:

- Requirements Specification
- Design Specification
- Unit Development Folders
- Test Plans and Procedures
- Test Results
- Maintenance Manuals

(Refs: 21:538-539; 16:44)

Vita

Michael Allen Tebo was born on 29 September 1953 in Bainbridge, Maryland to Annette and Jack Tebo. He graduated from Leon High School in Tallahassee in 1971 and continued his education at Florida State University, also in Tallahassee. His final two years were on an Air Force ROTC scholarship and he received a Bachelor of Science degree from the Mathematics Department in Computer Science in 1977. Through the ROTC program he was commissioned into the USAF upon graduation. His first assignment was to Wright-Patterson AFB where he worked as a computer systems analyst for the Computer Aided Design Facility in the Microelectronic Technology Branch of the Avionics Laboratory. After three and a half years he received his second assignment and entered the Air Force Institute of Technology, School of Engineering in June 1981.

Permanent Address: 3040 Watterford Drive
Tallahassee, Florida 32308

END

FILMED

3-83

DTIC